TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Vizualizace dat 3D prostorového senzoru v embedded systému

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Lukáš Sedláček**
Vedoucí práce: Ing. Martin Kysela

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Visualization of the 3D Spatial Sensor Data in Embedded System

Master thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Lukáš Sedláček**
Supervisor: Ing. Martin Kysela

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Lukáš Sedláček**
Osobní číslo:	**M15000242**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Vizualizace dat 3D prostorového senzoru v embedded systému**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s předloženým embedded systémem s OS Linux a HW řešením modulu a LCD, s 3D prostorovým senzorem využívaným pro rehabilitaci a formátem dat.
2. Nastudujte způsoby zobrazení 3D prostorových souřadnic na LCD a SW metody vykreslování v C/C++ a vyberte nejvhodnější.
3. Jednu zvolenou metodu 3D vizualizace naprogramujte do embedded zařízení tak, aby bylo možné SW dále rozšiřovat. Připravte obecné GUI s uživatelskými vstupy (klávesnice, myš, touch panel) spustitelné na emb. systému s OS Linux.
4. Otestujte zhotovený SW v praxi a zdokumentujte zdrojové kódy.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **40–50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Kadlec V. Učíme se programovat v jazyce C. Computer Press, 2002. ISBN: 80- 7226-715-9.**

[2] **ELIŠKA, Oldřich, Miloslava ELIŠKOVÁ. Aplikovaná anatomie pro fyzioterapeuty a maséry: hodnocení a léčba myofasciálních poruch. Vyd. 1. Praha: Galén, 2009, viii, 201 s. ISBN 978-802-4617-169.**

[3] **KEMPE, Volker. Inertial MEMS: principles and practice. New York: Cambridge University Press, 2011, xiv, 475 p. ISBN 05-217-6658-3.**

Vedoucí diplomové práce: **Ing. Martin Kysela**
Ústav mechatroniky a technické informatiky

Konzultant diplomové práce: **Ing. Matěj Kolář**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2016**

Termín odevzdání diplomové práce: **15. května 2017**

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2016

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 12. 5. 2017

Podpis:

Abstrakt

Diplomová práce se zaměřuje na problém vizualizace dat 3D prostorového senzoru s použitím vestavěného systému. Poznatky získané během řešení práce se odvíjí od možného použití aplikace v odvětví rehabilitace. Hlavním tématem je analýza prostorových senzorů a zpracování jejich poskytovaných dat. Dále se práce zabývá způsoby vykreslení 3D souřadnic na vestavěném systému a obsahuje návrh vlastního řešení. Cílem práce je především návrh a implementace aplikace, která vykresluje data z prostorového senzoru v reálném čase bez výrazného zpoždění. Závěrem rešerše jsou zhodnoceny poznatky z testování vlastní aplikace a doporučení pro další rozvoj.

Klíčová slova: 3D vizualizace, prostorový senzor, vestavěný systém, fúze dat, komplementární filtr, rehabilitace, křížová kompilace, minipočítač

Abstract

The thesis is focused on a problem of data visualization of 3D motion sensor with use of an embedded system. The knowledge gained during the thesis solution is based on future possible use of an application in the filed of rehabilitation. The main topic is an analysis of motion sensors and processing of provided data. Thesis also deals with the possible solutions of rendering 3D coordinates on embedded device and contains an own solution for thesis's problem. The goal of the project is primarily a design and an implementation of the application which can visualize data from the motion sensor in real time and without any noticeable delay. The conclusion contains findings from application's tests and also a recommendation for future application development.

Keywords: 3D visualisation, motion sensor, embedded system, sensor fusion, complementary filter, rehabilitation, cross compilation, single board computer

Poděkování

Rád bych poděkoval především vedoucímu mé diplomové práce, panu Ing. Martinu Kyselovi, a konzultantovi, panu Ing. Matěji Kolářovi, za cenné rady, předané znalosti a především za čas, který mi věnovali při konzultacích. Také děkuji panu doc. Ing. Pavlu Rydlovi, Ph.D za předané znalosti v oblasti matematického zpracování dat z prostorových senzorů. Dále bych chtěl poděkovat všem, kteří mi věnovali alespoň kousek svého času při řešení diplomové práce a v neposlední řadě také své rodině za neustálou podporu při mém dlouholetém studiu.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

TUL	Technická univerzita v Liberci
CRLF	Carriage Return, Line Feed
PC	Personal Computer, osobní počítač
USB	Universal Serial Bus, sériová linka
ARM	Advanced RISC Machine, architektura
WiFi	Protokol bezdrátové sítě, nejčastěji internet
GPIO	General-Purpose Input/Output
LCD	Liquid Crystal Display, typ displeje
OpenGL	Open Graphics Library, grafická knihovna
DoF	Degrees Of Freedom, stupně volnosti
ASCII	American Standard Code for Information Interchange
IDE	Integrated Development Environment
FPS	Frames Per Second, snímky za sekundu
RGB	Red Green Blue, barevný prostor
HTML	HyperText Markup Language, značkovací jazyk pro tvorbu hypertextových dokumentů
BGR	Blue Green Red
SSH	Secure Shell, zabezpečené vzdálené připojení

1 Úvod

V oblasti moderních technologií dochází již několik let k neustálému rozvoji vestavěných systémů. Se zmíněnými systémy se setkáváme každý den – systémy jsou součástí digitálních hodinek, mobilních telefonů, televizí, apod. Mezi nejvlivnější obory, které udržují neustálý rozvoj vestavěných zařízení, patří automobilový, vojenský a zábavní průmysl. Ovšem vestavěné systémy se uplatňují i v jiných oborech, například v medicíně. Embedded systémy a zařízení mohou velice usnadnit práci při vyšetření a následné léčbě pacientů.

Konkrétně v oboru rehabilitace, na který je diplomová práce mírně zaměřena, vestavěné systémy umožňují monitorovat práci svalů a poskytovat lékařům důležité informace pro léčbu pacientů. Zároveň s použitím prostorových senzorů je možné navrhnout řešení, které usnadní lékařům práci při samotných rehabilitacích – výsledkem je stav, kdy pacient rehabilituje sám s pomocí vestavěného systému bez nutné přítomnosti lékaře. Navíc by řešení poskytovalo vyhodnocení rehabilitace a případné návrhy pro další léčbu. V diplomové práci je řešena pouze vizualizace dat prostorového senzoru.

Tématem práce je především analýza poskytnutých minipočítačů, prostorových senzorů a zpracováním dat ze senzorů. Jelikož je cílem práce vytvořit pouze prototyp aplikace pro vizualizaci dat 3D prostorového senzoru, není navržen vlastní embedded systém, avšak bude použit běžně dostupný minipočítač. Kromě analýzy je dále cílem navrhnout vlastní řešení zmíněného problému. Řešení musí reagovat v reálném čase a bez viditelných prodlev. Literární rešerše se zabývá samotnou analýzou a návrhem řešení zadaného problému (zároveň popisuje důležitá rozhodnutí a problémy, které během práce nastaly). Praktická část je poté zaměřena na implementaci prototypu aplikace, která řeší vizualizaci dat prostorového senzoru na vestavěném systému.

2 Analýza

Prvním krokem při tvorbě nových aplikací je vždy analýza problému. Správně provedená analýza předchází v pozdějším vývoji obtížně řešitelným problémům či jiným rizikům (například bezpečnost vyvíjeného software), které mohou při následném návrhu, implementaci a použití nastat. Zároveň se analýza zabývá již dostupnými řešeními a hodnotí jejich možný přínos v řešení problému.

Během analýzy současných řešení vyšlo najevo, že aplikace podobného charakteru neexistuje či není volně dostupná. K dispozici jsou především řešení amatérských programátorů, avšak žádné neumožňuje vykreslit složitější 3D vizualizaci na embedded systému. Zároveň nebylo nalezeno řešení, které by bylo multiplatformní a které by používalo moderní technologie (knihovny) pro uživatelské rozhraní a 3D vizualizaci.

Stěžejním bodem diplomové práce je embedded systém, na kterém bude aplikace navržena a implementována. Systém je nutné zvolit tak, aby umožnil vykreslování 3D souřadnic a připojení prostorového senzoru. Dalším neméně podstatným bodem analýzy je výběr samotného prostorového senzoru a v neposlední řadě také matematické zpracování poskytovaných dat.

Kapitola se zabývá pouze zařízeními, které byly během diplomové práce k dispozici. Nejprve jsou popsány minipočítače a prostorové senzory. Dále se text zabývá zpracováním dat ze senzorů. Konec kapitoly je věnován analýze grafických platforem.

2.1 Embedded systémy

V české literatuře se pojem embedded systém [1] označuje jako *vestavěný systém*. Jedná se o systémy, které jsou součástí jiných strojů (například televizí, digitálních kamer, kalkulaček, apod.). Vestavěné systémy jsou určené pro řešení jedné specifické úlohy, nebo množiny speficických úloh. Jelikož jsou zařízení používána jednoúčelově, je možné je optimalizovat ke konkrétnímu použití a tím snížit jejich cenu. Základní myšlenka vestavěných systémů je, že jsou pro běžného uživatele skryté (neboli uživatel nepřijde do styku se samotným vestavěným systémem ale pouze s „hostujícím zařízením“). Embedded systémy jsou v dnešní době velice rozšířená zařízení [2]. Používají se například v automobilovém průmyslu, digitálních hodinkách, semaforech, MP3 přehrávačích, tepelných regulacích, apod.

Návrh jednoúčelového embedded systému není předmětem diplomové práce. Po

domluvě s vedoucím diplomové práce bude v projektu použit minipočítač[1] [3], který bude simulovat činnost embedded systému (tzn. minipočítač bude řešit pouze problém vizualizace dat 3D prostorového senzoru). Podobný princip použití minipočítače se často aplikuje i v praxi, pokud je potřebné levně ověřit navrhnuté hypotézy, protože tvorba jednoho specifického prototypu vestavěného systému je drahá.

Existuje velké množství minipočítačů [4], které jsou na trhu běžně dostupné. K důkladnější analýze byly dostupné tři minipočítače, které jsou popsány v následujících podsekcích. Během analýzy systémů bylo nutné se zaměřit zejména na možnosti vykreslení 3D souřadnic a na konektivitu periférií pro připojení prostorového senzoru.

2.1.1 Acqua A5

Acqua A5 je deska italského výrobce Acme systems [5]. K otestování byla dodána s nadstavbou Berta A5 LCD [6] – rozšiřujícím modulem pro připojení LCD displeje a samotným dotykovým displejem. Na zařízení je možné používat základní linuxový operační systém; v práci byl konkrétně v odlehčené distribuci Debian [7]. Připojení periférií k minipočítači je možné pomocí GPIO pinů (s podporou I^2C, SPI, UART,…), USB portů či ethernetového konektoru. Berta A5 lze také rozšířit o WiFi modul.

Během testování zařízení byly detekovány problémy s přiloženým diplejem. Barvy byly nepřesné – problém odpovídá použití barevného prostoru BGR namísto klasického RGB. Dále se vyskytly problémy se samotným dotykem. Souřadnice dotyku byly zatížené odchylkou v řádech několika centrimetrů. Zmíněné problémy se nepodařilo během testování desky odstranit. Příčinou mohl být vadný displej či špatně použité systémové ovladače (charakter chyb odpovídá spíše špatným ovladačům).

Největším problémem analyzovaného minipočítače je ovšem skutečnost, že neobsahuje grafický čip. Při vykreslení obrazu je tak nutné spoléhat pouze na samotný procesor. Zároveň je tím omezena i podpora grafické knihovny OpenGL [8], která na přítomnost grafického čipu spoléhá. Avšak existují implementace OpenGL, které lze používat i bez grafického čipu (například Mesa3D). Alternativní implementace ovšem spoléhají na větší výkon procesoru, jelikož procesor nemusí mít speciální instrukce pro zpracování obrazu a musí je simulovat.

2.1.2 Raspberry Pi B+

Raspberry Pi je minipočítač velikosti kreditní karty od společnosti Raspberry Pi Foundation. Zařízení bylo vyvinuto s cílem podpořit výuku informačních technologií na školách a v rozvojových zemích. Konkrétně verze B+ je již čtvrtou verzí známého minipočítače [9].

Hardwarově i softwarově je minipočítač podporován z oficiálních i neoficiálních zdrojů. Existuje nepřeberné množství hardwarových rozšíření pro Raspberry Pi (displeje, zvukové karty,…). Primárním operačním systémem je *Raspbian*, který je

[1] Anglicky *single-board computer*

oficální linuxou distribucí od mateřské společnosti. *Raspbian* je k dispozici s grafickou nadstavbou či bez grafické nadstavby (pouze příkazový řádek).

Na rozdíl od desky Acqua A5 2.1.1 nebyl k testování dodán displej. Raspberry Pi má ovšem možnost grafického výstupu přes rozhraní HDMI. Další konektivita periférií je zprostředkována přes GPIO piny (I^2C, SPI, UART,...), USB, RCA a ethernetový kabel.

Z hlediska vykreslování 3D souřadnic vyniká Raspberry Pi B+ grafickým čipem VideoCore IV. Paměť čipu je sdílena s operační pamětí. Zmíněný čip také podporuje OpenGL ve verzi ES 2.0 – odlehčená verze klasického desktopového OpenGL rozhraní, která se používá pro embedded systémy, minipočítače, chytrá zařízení, herní konzole, apod. Z předchozího textu je zřejmé, že vykreslení složitější vizualizace není na Raspberry Pi problémové. Například při testování modulu Canvas3D knihovny Qt 3.3 dosahovaly ukázkové příklady 30-50 FPS, což je pro plynulou vizualizaci dostatečné.

2.1.3 Raspberry Pi 3

Nejnovější a v současnosti nejvýkonější verze minipočítače Raspberry Pi je 3B [10]. Oproti Raspberry Pi B+ se kromě výkonu liší zejména v nově obsažené Bluetooth a WiFi konektivitě, která je velkou výhodou pro připojení periférií či pro samotnou komunikaci se zařízením. Zařízení je také prvním výrobkem společnosti Raspberry Pi Foundation, které obsahuje 64bitový procesor.

2.1.4 Porovnání zařízení

Z důvodu jednoduššího přehledu popisovaných zařízení byla vytvořena tabulka porovnání minipočítačů 2.1, které byly v předešlých kapitolách popsány. Tabulka obsahuje pouze rozhodující kritéria pro výběr správného zařízení v rámci diplomové práce.

2.2 Prostorový senzor

Dalším neméně důležitým předmětem analýzy je prostorový senzor. Prostorovým senzorem rozumíme řešení, které na základě svých interních čidel umožňuje určit svou orientaci v prostoru. Prostorový senzor obsahuje minimálně jedno z čidel akcelerometr, magnetometr nebo gyroskop. Každé ze zmíněných čidel může být digitální či analogové a zároveň každé může být jednoosé, dvouosé nebo tříosé.

Použití senzoru s pouze jedním interním čidlem nemusí být dostatečné, jelikož každé zmíněné čidlo je zatíženo vlastní chybou měření. Kombinací více čidel lze orientaci prostorového senzoru získat za pomoci algoritmu fúze dat [11] a tím minimalizovat chybu měření. To znamená, že chybu měření jednoho čidla lze do určité míry (závisí na nastavení algoritmu) kompenzovat jiným čidlem.

Obecně lze pomocí prostorového senzoru určit směr pohybu (a samozřejmě výslednou novou polohu):

Tabulka 2.1: Porovnání minipočítačů

	Acqua A5 (Berta A5)	Raspberry Pi B+	Raspberry Pi 3
Procesor	Atmel MPU SAMA5D31	Broadcom BCM2835	64bit ARM Cortex-A53
Operační paměť	256 MB DDR2	512 MB SDRAM (sdíleno s GPU)	1 GB SDRAM (sdíleno GPU)
Grafický čip	Neobsahuje	VideoCore IV dual-core GPU	VideoCore IV dual-core GPU
Konektivita	Ethernet, 2×USB, GPIO (použití jako I^2C, SPI, UART, …), možnost rozšíření o Wifi modul	Ethernet, 4×USB 3,5 mm RCA, HDMI 1.3, GPIO (použití jako I^2C, SPI, UART …)	Ethernet, WiFi, Bluetooth, 4×USB, 3,5 mm RCA, HDMI 1.3, GPIO (použití jako I^2C, SPI, UART …)
verze OpenGL	—	OpenGL ES 2.0	OpenGL ES 2.0, experimentálně Desktop OpenGL

- nahoru/dolů
- doleva/doprava
- dopředu/dozadu

avšak je nutné, aby senzor obsahoval čidlo akcelerometru. Dále je možné určit také rotaci senzoru, často popisovanou pomocí Eulerových úhlů [12] (anglické ekvivalenty úhlů jsou uvedené v závorce):

- okolo osy X (roll)
- okolo osy Y (pitch)
- okolo osy Z (yaw)

Stejně jako u embedded systémů 2.1 platí, že existuje velké množství prostorových senzorů, které se liší především svou citlivostí a počtem a typem obsažených čidel. Předmětem analýzy byly dva dostupné senzory – ukazovátko GiTy a senzor LSM9DS0.

2.2.1 GiTy

Prvním analyzovaným senzorem v diplomové práci bylo ukazovátko GiTy (obrázek 2.1). Zařízení je vyrobeno českou stejnojmennou firmou, která se primárně zabývá telekomunikací [13]. Původně mělo být ukazovátko použito pouze pro účely videokonference, avšak ve spolupráci s TUL je možné rozšířit využití ukazovátka o odvětví rehabilitace. Ukazovátko je již druhým prototypem prostorového senzoru od zmíněné společnosti. Oproti první verzi je nově možné senzor bezdrátově nabíjet. Přínosný je dále také moderní ergonomičtější vzhled [14].

Obrázek 2.1: Prostorový senzor GiTy

Data jsou z ukazovátka získávána bezdrátovou technologií Bluetooth [15]. Tzn. že je možné senzor spárovat s jakýmkoli jiným cílovým zařízením, které podporuje stejnou technologii. Data jsou z GiTy odesílána v pravidelných intervalech (průměrem každých 25 ms), avšak může docházet k prodlevě při přenosou bezdrátovou komunikací – častější žádost, duplikát paketu či opravení chybných paketů samoopravným kódem. Telefonátem s vývojáři společnosti GiTy bylo ověřeno, že data poskytovaná ukazovátkem nejsou dále upravována. Dále bylo ověřeno, že akcelerometr má nastavenou citlivost ± 2 g.

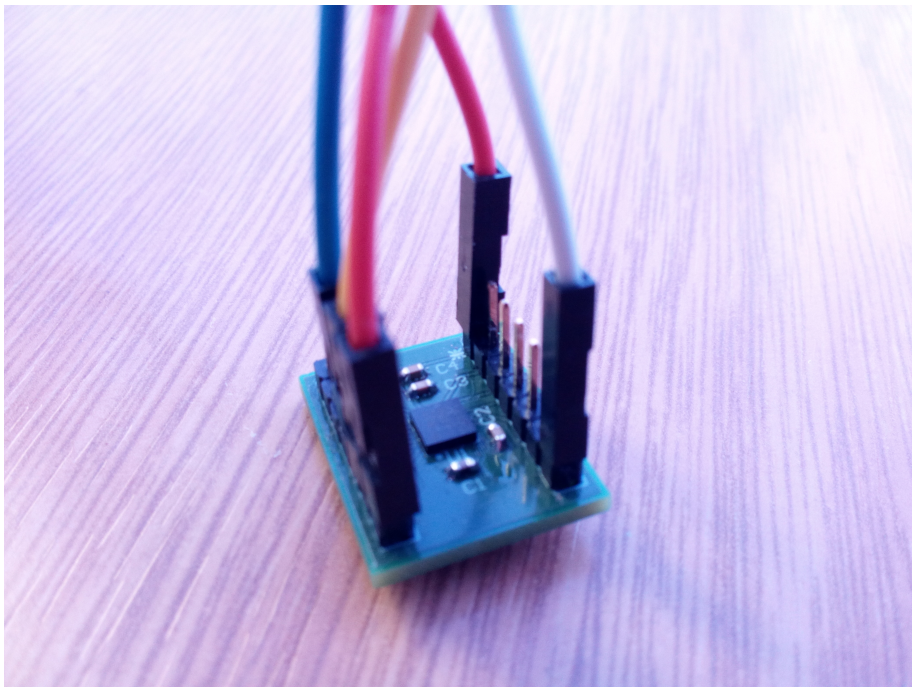
K ukazovátku neexistuje žádná dokumentace. Reverzním inženýrstvím[2] bylo zjištěno, že senzor obsahuje digitální tříosý akcelerometr ADXL345 [16] a digitální tříosý gyroskop ITG3200 [17]. Obsažená čidla dávají ukazovátku možnost pracovat až se šesti stupni volnosti (6 DoF). Z dat, která jsou poskytnutá jednotlivými čidly, lze dopočítat úhly *pitch* a *roll* 2.2. Úhel *yaw* lze získat pouze z gyroskopu, který je ovšem zatíženou chybou zvanou „Gyroskop drift“ 2.3.2.

Poskytovaná data Po spárování GiTy s cílovým zařízením a připojením k sériovému portu je možné číst data z ukazovátka. Data jsou v ASCII formátu. V následující ukázce je příklad jednoho získaného záznamu:

```
0,0,0,0,0,551.07,GYRO:6.42,8.11,-29.28,ACCEL:-6.26,-2.51,253.20
```

Z formátu dat je patrné, že se zde vyskytují surová data tříosého akcelerometru a gyroskopu. Kromě dat senzorů jsou v záznamu obsaženy informace o pohybu zařízení v ose X a Y, které jsou spočítané interním algoritmem z hodnot senzorů (algoritmus není znám) a dále informace o aktuálním stavu stisknutí třech hardwarových tlačítek a úrovni nabití baterie. Aby bylo možné jednotlivé záznamy rozlišit, je každý řádek ukončen sekvencí CRLF.

[2]Proces odhalení činnosti již fungujícího předmětu



Obrázek 2.2: Prostorový senzor LSM9DS0

2.2.2 LSM9DS0

Druhým analyzovaným prostorovým senzorem je LSM9DS0 [18]. Zařízení je běžně dostupným senzorem od společnosti STMicroelectronics. Na rozdíl od GiTy, které je již hotovým řešením zabaleným do ergonomického pouzdra, je LSM9DS0 pouze samotný prostorový senzor (obrázek 2.2).

Komunikace se zařízením je založena na sběrnici I^2C [19]. Vedoucí diplomové práce poskytl senzor s již hotovou redukcí pro snadné připojení k GPIO pinům minipočítače. Senzor je napájen ze zdroje poskytujícho 3,3 V.

K prostorovému senzoru je dostupná základní dokumentace a popis jeho vlastností. Senzor obsahuje tři digitální tříosá čidla – akcelerometr, magnetometr, gyroskop – tudíž je možné pracovat až s devíti stupni volnosti (9 DoF). Oproti předchozímu popisovanému senzoru 2.2.1 je možné navíc správně kompenzovat úhel *yaw* a tím získat přesné rotace okolo všech tří os.

Poskytovaná data – aby bylo možné vyčíst data ze senzoru, je nutné na hostitelském stroji zavést vhodné moduly (ovladače) do linuxového kernelu. Moduly jsou volně dostupné [20] a kompilují se ze zdrojových souborů. Ovladače krom jiného nastavují rychlost vyčítání dat a citlivost jednotlivých čidel.

Na rozdíl od GiTy jsou data v binárním formátu a tudíž nelze uvést příklad konkrétního získaného záznamu. Binární data jsou v C/C++ jazyce vyčtena přímo do struktury *input_event*, která je součástí zdrojových kódu linuxového jádra. Další zpracování dat se provádí přímo nad zmíněnou strukturou.

Jednotlivá čidla senzoru jsou velice přesná a stabilní. Obsažený šum je minimální.

2.3 Zpracování dat

Práce se zabývá vizualizací dat 3D prostorového senzoru. Konkrétně se práce zaměřuje na zobrazení orientace prostorového senzoru, neboli jeho natočení v prostoru. Pro vhodné vykreslení dat je nutné surová data převést do vhodného matematického modelu – ve 3D vizualizacích se používají Eulerovy úhly (sekce 2.2).

2.3.1 Akcelerometr

Data z akcelerometru reprezentují gravitační zrychlení působící ve směru jednotlivých os. Složením sil, které působí v jednotlivých osách, je možné získat směr pohybu zařízení a případně také natočení senzoru v prostoru. Náklon senzoru lze u tříosého akcelerometru určit pouze v úhlech *roll* a *pitch* (Eulerovy úhly, sekce 2.2). Rotaci okolo osy z (úhel *yaw*) nelze určit, jelikož při zmíněné rotaci jsou gravitační zrychlení v jednotlivých osách konstantní. Data akcelerometru se filtrují dolnopropustním filtrem [21], z důvodu vyhlazení odstranění šumu (obrázek 2.3). Jednoduchý filtr dolní propusti se implementuje dle vztahu 2.1, kde y jsou vypočtené hodnoty, x představuje aktuální surová data ze senzoru a $\alpha \in \langle 0, 1 \rangle$ určuje vliv flitru.

$$y[n] = \alpha \cdot x[n] + (1 - \alpha) \cdot y[n - 1] \tag{2.1}$$

Pro výpočet náklonu zařízení z dat akcelerometru se používají goniometrické funkce *sinus* a *cosinus*. Pokud označíme data z čidla jako a_x, a_y a a_z potom je možné spočítat náklon podle vztahů 2.2 a 2.3, které vycházejí z publikace od Marka Pedleyho [22].

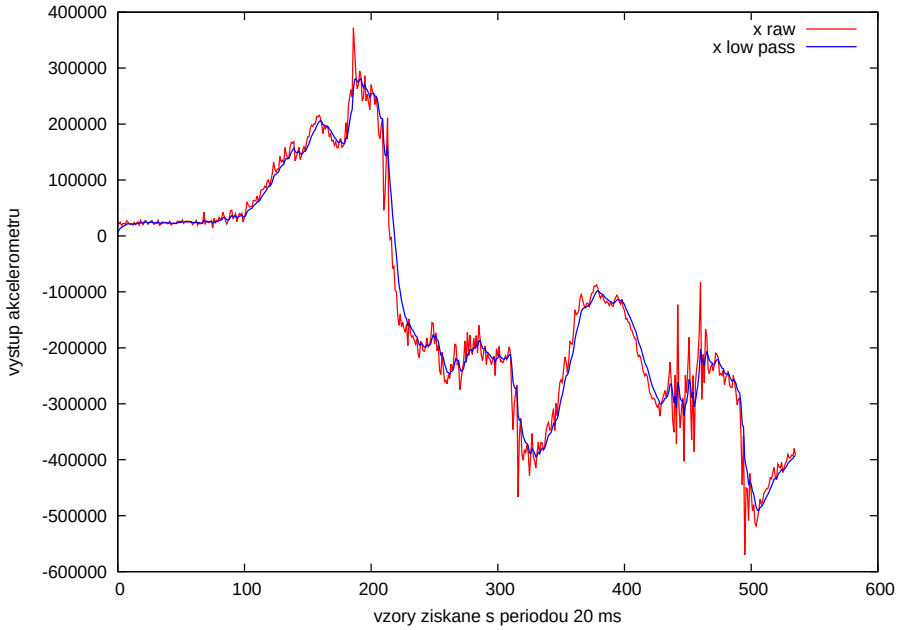
$$\varphi_{pitch} = arctg\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right) \tag{2.2}$$

$$\varphi_{roll} = arctg\left(\frac{-a_y}{\sqrt{a_x^2 + a_z^2}}\right) \tag{2.3}$$

Namísto klasické funkce *arctg* se častěji používá funkce *arctg2* [23], jejíž obor hodnot je definován jako interval $\langle 0, 2\pi)$. V programovacích jazycích je běžné, že je zmíněný interval posunut o hodnotu π – obor hodnot funkce *arctg2* je poté $\langle -\pi, \pi)$.

V rovnicích 2.2 a 2.3 se ve jmenovateli vyskytuje odmocnina ze součtu kvadrátů hodnot akcelerometru. Protože jmenovatel bude vždy kladný, obor hodnot funkce *arctg* bude omezen pouze na interval $\left\langle -\frac{\pi}{2}, \frac{\pi}{2}\right)$. Z důvodu rozšíření intervalu na obor hodnot *arctg2* je nutné provést úpravu 2.4 pro úhel φ_{pitch} a úpravu 2.5 pro úhel φ_{roll}. Přepočty vycházejí z konzultací s panem doc. Ing. Pavlem Rydlem, Ph.D.

$$\varphi_{pitch} = \begin{cases} -\pi - \varphi_{pitch}, & (a_z < 0) \wedge (a_x > 0) \\ \pi - \varphi_{pitch}, & (a_z < 0) \wedge (a_x < 0) \\ \varphi_{pitch}, & \text{jinde} \end{cases} \tag{2.4}$$



Obrázek 2.3: Dolní propusť akcelerometru senzoru LSM9DS0, $\alpha = 0,25$

$$\varphi_{roll} = \begin{cases} -\pi - \varphi_{roll}, & (a_z < 0) \wedge (a_y > 0) \\ \pi - \varphi_{roll}, & (a_z < 0) \wedge (a_y < 0) \\ \varphi_{roll}, & \text{jinde} \end{cases} \tag{2.5}$$

2.3.2 Gyroskop

Data gyroskopu reprezentují úhlové rychlosti při rotaci senzoru okolo jednotlivých os. Data jsou v jednotkách $deg \cdot s^{-1}$. Jelikož gyroskop zaznamenává pouze změny oproti předchozímu stavu, filtrují se data pomocí horní propusti, aby v klidovém stavu nedocházelo k odchylce z důvodu zašuměných dat. Jednoduchý hornopropustní filtr [21] se implementuje podle vztahu 2.6, kde y jsou počítané hodnoty, x jsou surová data ze senzoru a $\alpha \in \langle 0, 1 \rangle$ určuje vliv flitru. Dále uvedené vzorce vycházejí z reference [24].

$$y[n] = \alpha \cdot (y[n-1] + x[n] - x[n-1]) \tag{2.6}$$

Úhlová rychlost ω je dána vztahem 2.7.

$$\omega = \frac{d\varphi}{dt} \tag{2.7}$$

Aby bylo možné spočítat úhel φ, o který se zařízení oproti předchozímu stavu za čas dt otočilo, je nutné rovnici 2.7 integrovat podle času 2.8.

$$\varphi = \int_0^t \omega dt \approx \sum_0^t \omega(t) \tag{2.8}$$

Pokud budeme uvažovat, že hodnota $\omega(t)$ je po celou dobu změny t konstatní, je možné rovnici 2.8 zjednodušit na tvar 2.9

$$\varphi = \omega \cdot t \tag{2.9}$$

Hodnota ω představuje surová data načtená z gyroskopu v jednotkách $deg \cdot s^{-1}$ a hodnota t představuje interval mezi jednotlivými záznamy (ve zdrojových kódech se označuje jako „poll interval“).

Pokud bychom určovali natočení prostorového senzoru pouze z hodnot gyroskopu, bude nutné akumulovat jednotlivě získané změny φ. Jelikož jsou změny spočítány aproximací dle rovnic 2.8 a 2.9, bude se neustále zvětšovat chyba měření. Defekt je obecně známý a nazývá se „Gyroskop drift“ [25]. Neboli při reálném otočení senzoru o úhel 2π bude akumulovaný úhel větší jak 2π.

2.3.3 Magnetometr

Magnetometr je senzor měřící sílu magnetického pole. Data poskytovaná ze senzoru udávají velikost magnetického pole v jednotlivých osách. Hodnoty jsou v jednotkách *gauss*. Z dat magnetometru je možné dopočítat výsledný směr magnetického pole, tzn. že je možné senzor používat jako kompas. Neboli data magnetometru jsou vhodná k určení rotace okolo osy Z – uhel *yaw*. Podobně jako u dat z akcelerometru (sekce 2.3.1) je nutné vyhladit záznamy pomocí dolnopropustního filtru (rovnice 2.1).

Azimut φ_{azimut} (úhel odchýlení od severu) prostorového senzoru je možné spočítat pomocí rovnice 2.10, kde hodnoty m_x, m_y, m_z jsou surová data magnetometru.

$$\varphi_{azimut} = arctg\left(\frac{-m_y}{m_x}\right) \tag{2.10}$$

Rovnice 2.10 ovšem platí pouze pokud je prostorový senzor umístěn na vodorovné podložce. Jakmile dojde k náklonu zařízení, změní se působení magnetických sil v jednotlivých osách a určení azimutu φ_{azimut} nebude přesné [26].

2.3.4 Fúze dat

Každé čidlo v prostorovém senzoru je zatíženo určitým defektem. Jestliže senzor obsahuje minimálně dvě rozdílná čidla z množiny akcelerometr, magnetometr a gyroskop, je poté možné v některých úhlech kompenzovat defekt jednoho čidla jiným obsaženým čidlem. V následujícím textu je uveden způsob kompenzace defektů u senzoru, který obsahuje všechna zmíněná čidla. V případě menšího počtu čidel je postup analogický, avšak není poté možné kompenzovat všechny defekty prostorového senzoru.

Náklon magnetometru

Při popisu magnetometru v sekci 2.3.3 bylo konstatováno, že samotný vypočtený azimut je přesný pouze pokud je senzor umístěn na vodorovné podložce. Tento defekt

lze kompenzovat pomocí dat z akcelerometru. Jestliže máme data z akcelerometru a_x, a_y, a_z a data z magnetometru m_x, m_y, m_z, je možné přepočítat hodnoty magnetometru dle náklonů určených z dat akcelerometru. Výpočty 2.11 vycházejí z publikace [26].

$$\begin{aligned} \varphi_x &= asin(a_x) \\ \varphi_y &= -asin\left(\frac{a_y}{cos(\varphi_x)}\right) \\ mag_x &= m_x * cos(\varphi_x) + m_z * sin(\varphi_x) \\ mag_y &= m_x * sin(\varphi_y) * sin(\varphi_x) \\ &\quad + m_y * cos(\varphi_y) \\ &\quad - m_z * sin(\varphi_y) * cos(\varphi_x) \end{aligned} \tag{2.11}$$

Úhly φ_x a φ_y určují náklony senzoru a hodnoty mag_x a mag_y jsou nově vypočtené hodnoty magnetometru. Pro výpočet kompenzovaného azimutu je nutné dosadit mag_x a mag_y do rovnice 2.10.

Vyhlazení dat akcelerometru a magnetometru

I přes to, že jsou data z akcelerometru a magnetometru vyhlazena pomocí dolnopropustního filtru dle vzorce 2.1, jsou záznamy stále mírně zatížené šumem. Naproti tomu data z gyroskopu jsou stabilní, avšak při použití samotného gyroskopu dochází k defektu „Gyroskop drift“ 2.3.2.

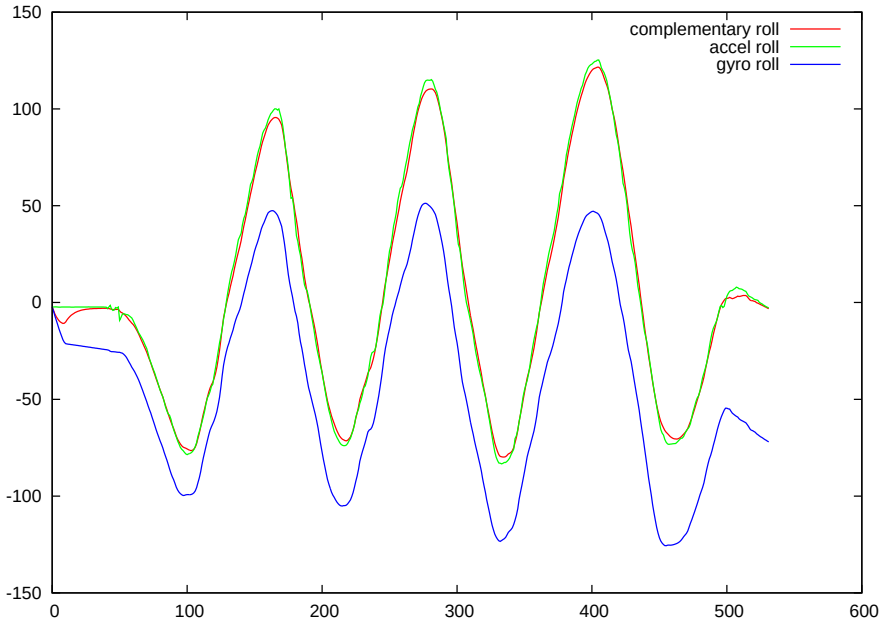
Aby se odstranily defekty jednotlivých senzorů, je možné implementovat algoritmus pro fúzi dat – především Kalmánův [27] či komplementární filtr[3] [28]. Kalmánův filtr poskytuje silnější model pro fúzi dat jednotlivých čidel, avšak je náročnější na pochopení a především na výpočetní prostředky. Z hlediska zachování jednoduchosti aplikace a s přihlédnutím k faktu, že aplikaci je nutné spouštět na embedded systému 2.1, je vhodné použít jednodušší komplementární filtr.

Komplementární filtr opravuje hodnoty gyroskopu o úhly spočtené z akcelerometru (při rotaci *roll* a *pitch*) a magnetometu (při rotaci *yaw*). Jestliže je φ_{sensor} úhel získaný z akcelerometru či magnetometru, $\Delta\varphi_{gyro}$ aktuální úhlová změna zaznamenaná gyroskopem, y počítané hodnoty a $\alpha \in \langle 0, 1\rangle$, je možné zapsat filtr dle rovnice 2.12.

$$y[n] = (1 - \alpha) \cdot (y[n-1] + \Delta\varphi_{gyro}) + \alpha \cdot \varphi_{sensor} \tag{2.12}$$

Hodnota α určuje poměr mezi úhlem $(y[n-1]+\Delta\varphi_{gyro})$ určeným relativní změnou gyroskopu oproti předchozí poloze a absolutním úhlem φ_{sensor} spočteným z jednoho ze zbylých senzorů. Neboli lze říci, že hodnoty z magnetometru a akcelerometru α-násobně opravují tzv. „Gyroskop drift“. Na obrázku 2.4 jsou zaznamenány vypočtené hodnoty úhlu *roll*, kde hodnoty jsou nejprve spočítány ze samotného akcelerometru a gyroskopu a následně jsou určeny fúzí dat zmíněných senzorů. K zaznamenání dat byl použit senzor LSM9DS0 2.2.

[3] angl. *Complementary filter*



Obrázek 2.4: Komplementární filtr s akcelerometrem, $\alpha = 0,15$

Přepočítané hodnoty z akcelerometru 2.3.1, respektive z magnetometru 2.3.3, jsou spočítané pomocí goniometrické funkce *arctg2*. Znamená to, že v závislosti na znaménkách hodnot zmíněných čidel dochází ke skokovému přechodu mezi úhly π a $-\pi$ (inkrementací úhlu π vznikne úhel $-\pi$). Komplementární filtr očekává dle změn gyroskopu 2.3.2, že hodnoty budou neustále růst, resp. klesat – jakmile dojde ke zmíněnému skokovému přechodu, začne se filtr dorovnávat dle hodnot akcelerometru, resp. magnetometru a naruší se stabilita filtru, který se po určitý čas zpětně srovnává do stabilní polohy. Aby se zachovala stabilita filtru i při přechodu, je nutné použít algoritmus popsaný v definici 2.13. Algoritmus je navržen tak, aby byl funkční jak pro výpočet *pitch* a *roll* závislých na hodnotách akcelerometru, tak pro výpočet *yaw*, který je závislý na hodnotách magnetometru. Úhel φ je původní výstup z komplementárního filtru, φ_{sensor} je předešle popsaný absolutní úhel a úhel $\alpha \in (-\pi, \pi)$ je hraniční úhel (určuje úhlovou hranici, od které je možné přepočet aplikovat).

$$\varphi = \begin{cases} 2\pi + \varphi, & (\varphi < -\alpha) \wedge (\varphi_{sensor} > \alpha) \\ -2\pi + \varphi, & (\varphi > \alpha) \wedge (\varphi_{sensor} < -\alpha) \\ \varphi, & \text{jinde} \end{cases} \tag{2.13}$$

2.4 Grafické platformy

Aplikace diplomové práce je zaměřena na práci s embedded systémem (sekce 2.1) a tudíž také na operační systém Linux, který se na vestavěných systémech používá nejčastěji. Linux poskytuje pouze samotné jádro operačního systému, nikoli grafickou nadstavbu, která je viditelná pro běžného uživatele. Existuje více platforem, které

umožňují vykreslit uživatelské rozhraní (například LinuxFB, EGLFS, XCB, Wayland,…). Výběrem vhodné grafické platformy je možné snížit použité prostředky na minimum (za cenu omezených možností vizualizace či složitější implementace). Zároveň se použití konkrétní platformy liší v závilosti na způsobu vykreslování – některé ze zmíněných plaforem umožňují oba typy vykreslení v závilosti na konfiguraci:

- Hardwarové – použití grafického čipu, který zajišťuje vykreslení obrazu
- Softwarové – vykreslování obrazu je zajištěno samotným procesorem zařízení

Následující text vychází z dokumentace ke knihovně Qt [29]. Ačkoliv je text zaměřen především na samotnou knihovnu, udává dokumentace přehled hlavních grafických platforem používaných na Linuxu a především na vestavěných systémech.

X11 poskytuje rozhraní pro desktopové uživatelské rozhraní. Společně s platformou *Wayland* tvoří již hotové řešení pro správce oken. Open-source implementace správce oken *X11* se nazývá X.Org.

Platforma se skládá ze tří hlavních komponent – zobrazovací server, klient a kompozitor. Server poskytuje rozhraní pro připojení jednotlivých klientů (aplikace s uživatelským rozhraním) a dále komunikuje s kompozitorem, který určuje způsob výsledného vykreslení aplikací do uživatelského prostředí. K implementaci klientů je k dispozici knihovna Xlib, která vytváří jednodušší aplikační rozhraní pro komunikaci se serverem.

Nevýhodou *X11* jsou požadavky na prostředky systému. Pří testování s minipočítačem Acqua A5 2.1.1 bylo zjištěno, že pouze samotný server rozhraní zabírá 20 MB paměti. Při návrhu embedded systému by tedy bylo nutné zajistit větší paměťovou kapacitu – tím se ovšem bude zvyšovat cena zařízení. Právě z hlediska paměťové náročnosti se platforma na embedded systémech nepoužívá, zde je uvedena především kvůli své rozšířenosti na Linuxu.

Wayland je odlehčený a jednodušší správce oken pro operační systémy Linux. Na rozdíl od přechozí popisované platformy má *Wayland* spojenou komponentu serveru a kompozitoru. V referenci [30] je možné porovnat návrhy platforem *X11* a *Wayland*. Z popisuje zřejmé, že *Wayland* šetří důležité prostředky, jelikož nemusí navíc komunikovat s komponentou kompozitoru. Pro zachování obecnosti je možné k serveru připojit klienta z předchozí popisované platformy.

Z hlediska použití je již popisovaná platforma vhodná pro embedded systémy, zejména poté v případě, že je nutné vykreslit více klientů najednou (více uživatelských oken). Implementace jednoduchého kompozitoru, který je vhodný pro vestavěná zařízení, se nazývá *Weston*. Podle oficiálních zdrojů k platformě bude v budoucnu v desktopových uživatelských rozhraních GNOME a KDE nahrazena závilost na platformě *X11* platformou *Wayland* z důvodu modernějšího a méně paměťově náročného přístupu.

LinuxFB patří mezi platformy, které zapisují přímo do tzv. „framebufferu“ – zjednodušeně jde o dvourozměrné pole bodů (velikosti šířka×výška obrazovky) obsahující jednotlivé vykreslované body obrazu. Předchozí dvě popisované platformy mají implementace pro hardwarové i softwarové vykreslování. *LinuxFB* umožňuje vykreslovat pouze softwarově, což je omezující především u 3D vizualizace. Knihovny pro 3D vizualizaci spoléhají na vykreslování pomocí grafického čipu (existují i alternativy, které vykreslují softwarově, avšak výkon procesoru musí být větší).

Při testování minipočítače Acqua A5 2.1.1 byla platforma *LinuxFB* používána s knihovnou Qt 3.3. Výkon procesoru byl dostatečný pro vykreslování jednodušších 2D vizualizací. Zmíněná knihovna umí vykreslovat i složitější kontext softwarově, avšak výkon procesoru nebyl dostatečný, jak bude dále popsáno v kapitole 4.

Dále je nutné podotknout, že samotná platforma neumožňuje zpracování uživatelských vstupů (především myš a klávesnice) a je nutné použít jinou podpůrnou knihovnu – například *libinput*.

EGLFS vychází z platformy *EGL*, která vytváří rozhraní mezi OpenGL a nativním správcem oken, který je dostupný na operačním systému (na Linuxu *X11* nebo *Wayland*). Platforma *EGLFS* zajišťuje vykreslování výstupů knihovny OpenGL bez nutnosti použití správce oken, tudíž šetří cenné hardwarové prostředky, které by byly využity pro běh správce.

Platforma zajišťuje softwarové i hardwarové vykreslování. Obsah, který je zpracován softwarově, je nejprve za pomocí procesoru vykreslen do obrázků, které jsou následně zpracovány a zobrazeny jako uživatelské rozhraní. Z hlediska charakteru platformy je nutné, aby vždy hlavní okno (respektive kanvas uživatelského rozhraní) bylo vykresleno na celou obrazovku.

EGLFS je platforma, která je doporučená pro použití na vestavěných systémech z hlediska nízkých požadavků na prostředky, avšak při zachování možnosti vykreslení složitějších vizualizací (s možností použití OpenGL).

3 Návrh

Po analytické části v kapitole 2 je dalším navazujícím krokem při vývoji nového software návrh aplikace. Během návrhu jsou zhodnoceny konkrétní technologie (jazyk programu, dostupné ovladače a knihovny, formát konfiguračního souboru, apod.) a architektura, na které bude aplikace zprovozněna. Dále je vhodné rozmyslet rozdělení aplikace na samostatné menší funkční bloky, které řeší určitou část zadaného problému a jejich následné spojení ve funkční celek.

Kapitola se nejprve zabývá architekturou zařízení a křížovou kompilací. Dále se kapitola zaměřuje na použité programovací jazyky a s tím související zvolené programovací knihovny, které usnadňují některé implementační kroky. Konec kapitoly je věnován rozdělení aplikace na jednotlivé funkční bloky a zejména také návrhu obecného programovacího rozhraní.

Výsledkem návrhu je jasná představa o konkrétních použitých technologiích a architektuře. Další důležitá rozhodnotí, která se týkají výběru prostorového senzoru a konkrétního minipočítače, jsou zhodnoceny v další kapitole 4.

3.1 Architektura

Embedded systémy 2.1, které byly k práci k dispozici, patří do rodiny zařízení s ARM architekturou. V posledních letech se ARM achitektura rychlým tempem rozšiřuje zejména kvůli stále větší oblíbenosti mobilních zařízení a tabletů [31], tudíž aplikace vytvořená pro zmíněnou architekturu může mít potenciálně velké rozšíření.

Při návrhu aplikace je ovšem nutné myslet na skutečnost, že minipočítače popsané v sekci 2.1 pouze simulují činnost embedded systému a byly použity pouze pro ověření základních hypotéz. Pokud by v dalším vývoji byla použita jiná architektura, bude vhodné aby bylo možné aplikaci bez větších obtíží zkompilovat pro jiné architektury. Zmíněný požadavek je nutné zohlednit při výběru knihoven 3.3.

Existuje velké množství dostupných operačních systémů pro vestavěná zařízení. Příkladem může být mbed, linux, RTOS, Windows Embedded, apod. Výběr konkrétního operačního systému závisí na použití embedded zařízení. Jelikož aplikace bude vyžadovat vláknové programování, dynamickou alokaci paměti a složitější 3D vizualizaci, je vhodné použít operační systém Linux.

Tabulka 3.1: Porovnání rychlosti kompilace

	Raspberry Pi 3+	Laptop Intel x64
„Hello World“ aplikace	1,6 s	0,133 s
Aplikace Huffmanova kódování (960 řádků)	17,379 s	1,219 s

3.1.1 Křížová kompilace

Křížovou kompilací[1] se rozumí kompilace aplikací či knihoven pro jinou architekturu, než je architektura kompilovacího stroje. Jelikož jsou vestavěné systémy jednoúčelové, jejich výkon je často omezený a kompilace pomalá. Křížová kompilace urychluje proces vývoje.

Porovnání rychlosti C++ kompilace nejvýkonnějšího minipočítače ze sekce 2.1 a běžně dostupného Intel x64 laptopu je znázorněno v tabulce 3.1. Z tabulky je patrné, že křížová kompilace je velice výhodná i při malých projektech. Čas byl měřen pomocí linuxového příkazu *time*.

Ke křížové kompilaci je nutné použít speciální kompilační nástroj (například tzv. „Linaro“). Nástroj umožňuje vytvořit binární soubor (či knihovnu), který je spustitelný na cílové architektuře. Aby bylo možné zkompilovat zdrojové kódy pro cílovou platformu, je nutné vytvořit tzv. „sysroot“, neboli systémový adresář. Sysroot obsahuje knihovny a hlavičkové soubory zkopírované z cílového zařízení (zkompilované pod cílovou architekturou) do kompilovacího stroje. Kompilátoru je nutné předat cestu k vytvořenému systémovému adresáři.

Při křížové kompilaci je také nutné používat verzi křížového kompilátoru, která je kompatibilní s verzí kompilátoru na cílovém stroji. Jestliže se verze liší, nemusí dojít k sestavení programu, nebo se mohou projevit neočekávané chyby – například u kompilátoru GCC nefunguje standardní třída pro vlákna *std::thread*.

3.2 Jazyk programu

Výběr programovacího jazyku se odvíjí především od povahy aplikace, která musí rychle a efektivně překreslovat 3D vizualizaci dle dat, která se vyčítají ze senzorů. Jelikož musí být aplikace dle požadavků spustitelná na embedded systému, je nutné zohlednit také omezenější prostředky. Z porovnání rychlosti a paměťové náročnosti současných programovacích jazyků [32] je patrné, že je vhodné používat kompilované jazyky, konkrétně jazyk C++. Pokud navíc porovnáme rychlost vyčtení dat z GPIO pinů [33], je zvolený jazyk správnou volbou.

Z důvodu zvolené knihovny 3.3 bude nutné dále použít jazyky QML a JavaScript. Zmíněné jazyky umožňují definovat moderní uživatelská rozhraní. Navíc jazyk QML poskytuje dobrou podporu pro responzivní design, tudíž automaticky mění vzhled aplikace při použití na menších rozlišeních či obrazovkách. Pro zachování rychlosti aplikace a obecnosti implementačního rozhraní budou zmíněné jazyky použity pouze

[1] angl. *Cross compilation*

pro uživatelské rozhraní, nikoli pro řešení logiky aplikace (logika bude implementována v C++). Popsaný moderní přístup propojení QML a JavaScriptu s C++ se používá například v linuxových grafických rozhraních Unity a KDE Plasma.

3.3 Dostupné knihovny

Společně s výběrem programovacího jazyku 3.2 se vybírají také knihovny. Kromě standardních C++ knihoven budou použity i volně dostupné nestandardní knihovny, které implementují potřebné funkcionality pro vykreslení 3D vizualizace a čtení dat ze zařízení.

Na linuxovém operačním systému není nutné používat speciální knihovny pro čtení dat ze senzorů 2.2. Každý senzor se zobrazuje ve filesystému jako soubor, ze kterého je možné data rovnou číst (uživatel musí být ve skupině *dialout*). Může se stát, že k vyčtení dat je potřebné doinstalovat ovladače senzoru, které se zkompilují a zavedou přímo do jádra operačního systému.

Pro zpracování konfiguračního souboru 3.4.3 ve formátu JSON bude použita open-source[2] knihovna [34], která je zvolená na základě předchozích zkušeností. Navíc je rozhraní zmíněné knihovny přizpůsobené rozhraním tříd z C++ standardní knihovny, tudíž je možné s knihovnou JSON používat i některé standardní funkce pro vyhledávání, mazání, řazení prvků, apod.

Jelikož je problém 3D vizualizace velice komplexní, bude v diplomové práci použita knihovna, která již daný problém řeší. Zároveň je možné použitím vhodné nestandardní knihovny docílit softwaru, který bude multiplatformní – bude možné jej bezproblémově zkompilovat pro jinou architekturu, případně pro jiný operační systém. Open-source knihovny které zaručují multiplatformní použití a podporují vykreslení 3D vizualizace jsou popsány v následujících odstavcích.

JUCE [35] je knihovna vyvíjena v současnosti společností ROLI Ltd. (originálním autorem je Jules Storer). Knihovna poskytuje moderní vzhled a jednoduché aplikační rozhraní pro psaní aplikací. Zkompilovat knihovnu je možné pro všechny operační systémy Windows, Mac, Linux a navíc ještě pro platformu Android. Součástí knihovny je také vlastní vývojové prostředí a detailní dokumentace.

Aplikace napsané v knihovně JUCE potřebují ke svému běhu správce oken. Na operačním systému Linux je tak vyžadována platforma X11, což z hlediska použitých prostředků není na embedded systémech vhodné 2.4. Podle informací, které na oficiálním diskuzním fóru projektu sdílel původní autor je možné, že v budoucnosti bude knihovna připravena i pro použití na vestavěných systémech s použitím přímého vykreslení do linuxového framebufferu.

GTK+ [36] patří mezi nejznámější knihovnu uživatelského rozhraní na operačních systémech Linux, jelikož je v ní napsáno grafické rozhraní GNOME. I když jsou aplikace napsané v knihovně primárně určené pro Linux, je možné je provozovat i

[2] Knihovny s otevřeným zdrojovým kódem

na alternativých operačních systémech. Navíc lze GTK+ využít i bez správce oken, pokud použijeme vhodnou platformu, která vykresluje obraz přímo do framebufferu 2.4.

Qt [37] není pouze knihovnou uživatelského rozhraní, ale poskytuje funkcionality vhodné k jednodušší práci s vlákny, datovými strukturami, apod. Knihovna je velice komplexní a existuje velké množství modulů, které ještě navíc rozšiřují základní funkcionalitu knihovny (práce s grafy, multimédii, senzory, apod.). Qt také poskytuje vlastní vývojové prostředí Qt Creator. Prostředí umožňuje snadno a rychle měnit achitekturu, pro kterou se daný kód kompiluje a navíc poskytuje mechanismy pro vzdálený vývoj pomocí SSH. Vzdálený vývoj je vhodný především při práci s embedded systémy či minipočítači. Zdrojový kód je zkompilován na hostujícím stroji a výsledný binární soubor je odeslán na cílové zařízení, kde je spuštěn.

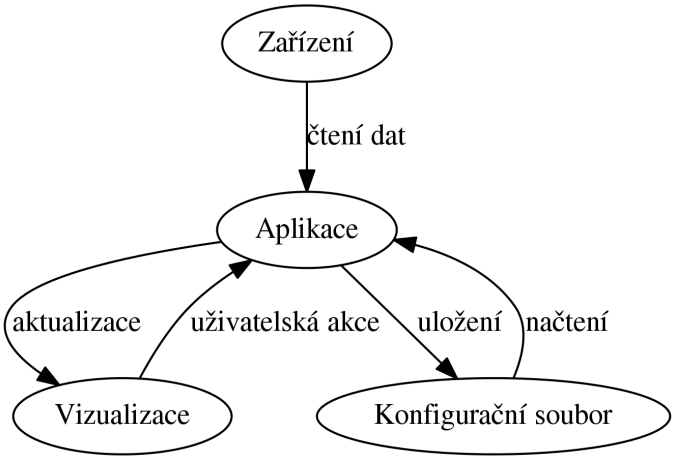
Knihovna je velice moderním přístupem ke grafickým rozhraním. V současné verzi využívá jazyky C++ a QML. Qt se také neustále vyvíjí – během diplomové práce se přešlo od verze 5.7 na verzi 5.9. S každou novou verzí se také postupně přesouvají moduly s proprietární licencí mezi open-source moduly.

Qt jako jediná z dostupných řešení poskytuje nativní podporu nejen pro počítače, ale také přímo pro embedded systémy 2.1. Podpora vestavěných systémů je vyřešena pomocí přepínatelných grafických platforem 2.4, které lze specifikovat až při spouštění aplikace. Navíc není nutné, aby vestavěné zařízení obsahovalo grafický čip – Qt od verze 5.8 obsahuje softwarový vykreslovač, který veškeré operace prováděné na grafickém čipu simuluje na procesoru (výkon procesoru musí ovšem být dostatečný).

Knihovnu je nutné zkompilovat pro konkrétní architekturu. Pro více známé architektury a zařízení (například Raspberry Pi 2.1.2) jsou nastavení kompilace již připravena k použití. Při migraci na jinou achitekturu je poté potřebné pouze překompilovat zdrojový kód pro cílový systém. Z důvodu vysoké podpory embedded systémů bude knihovna použita při vytváření prototypu aplikace diplomové práce.

K implementaci uživatelského rozhraní bude použit modul *quickcontrols2*, který v moderním stylu definuje ovládací prvky pomocí QML. Pro 3D Vizualizaci jsou dostupné dva moduly. Modul *3d* se nepodařilo na dostupných minipočítačích zprovoznit – docházelo k problémům se zápisem do grafické paměti. V diplomové práci bude použit modul *canvas3d*, který poskytuje rozhraní podobné WebGL pro zobrazení a práci s 3D objekty. Samotý kód 3D vizualizace je napsán v JavaScriptu za pomoci knihovny Three.js. Zmíněná knihovna je dostupná v knihovně Qt a poskytuje jednoduché rozhraní pro tvorbu a ovládání trojrozměrné vizualizace. Knihovna Three.js se běžně používá ve webových rozhraních ve WebGL – ve Qt je ovšem přizpůsobena pro použití s QML.

Všechny prvky moderních vizualizačních modulů pro Qt jsou také přizpůsobeny pro dotykové ovládání. Zároveň platí, že moderní moduly nejsou závislé na konkrétní architektuře, tudíž aplikace vypadají na všech platformách stejně.



Obrázek 3.1: Rozdělení aplikace

3.4 Rozdělení aplikace

Problém vizualizace dat 3D prostorového senzoru je komplexní problém. Pro zjednodušení implementace a zejména také pro větší přehlednost a obecnost zdrojových kódů je vhodné aplikaci rozdělit na tři základní funkční bloky:

- Vizualizace
- Práce se zařízením
- Konfigurační soubor

Rozdělení aplikace umožní jednoduše reprodukovat jednotlivé funkční bloky do jiných projektů. Zároveň bude implementace obecnější a kód bude možné lépe udržovat a rozšiřovat. Na obrázku 3.1 je zobrazeno propojení jednotlivých částí aplikace ve funkční celek. Je zřejmé, že aplikace bude mít vždy jednu centrální komponentu (*Aplikace*), která slouží k propojení ostatních funkčních bloků.

Návrhový vzor Propojení komponent na obrázku 3.1 připomíná návrhový vzor *Model-View-Controller* [38]. Vizualizaci lze chápat jako „View“, centrální blok jako „Controller“ a data ze zařízení jako „Model“. Na rozdíl od klasického pojetí popisovaného návrhové vzoru zde není možné do modelu data zapisovat, resp. model umožňuje pouze čtení dat ze senzoru na žádost centrálního řídicího bloku. Navíc se data v modelu asynchronně mění, jelikož čtení dat není pro senzor blokující. Neboli po připojení senzoru se data neustále vyčítají do souboru (který se následně čte) a funkční blok *Zařízení* získává pouze poslední načtenou hodnotu.

3.4.1 Vizualizace

Funkční blok *Vizualizace* umožňuje dvě základní funkce: aktualizaci uživatelského rozhraní a odchycení uživatelských akcí. Aktualizace rozhraní může probíhat synchronně (reakce na akci uživatele) a nebo asynchronně (po vyčtení dat ze zařízení se

rozhraní aktualizuje bez akce uživatele). Uživatelskými akcemi lze chápat jakýkoli vstup uživatele do aplikace pomocí myši, klávesnice, dotykového displeje, grafického tabletu, apod.

Vizualizace předává uživatelskou akci do centrálního řídicího prvku, který zpracuje konkrétní akci a může dle charakteru akce aktualizovat uživatelské rozhraní. Popisovaný funkční blok dále umožňuje zaregistrovat funkci pro asynchronní zpracování dat a aktualizaci grafického rozhraní, která musí být spuštěna v samostatném vlákně, aby nechodocházelo k zablokování uživatelského rozhraní při práci aplikace. Z důvodu vícevláknové aplikace je nutné, aby třídy, které zasahují do dat vizualizace, byly implementovány jako „thread safe“ třídy.

3.4.2 Zařízení

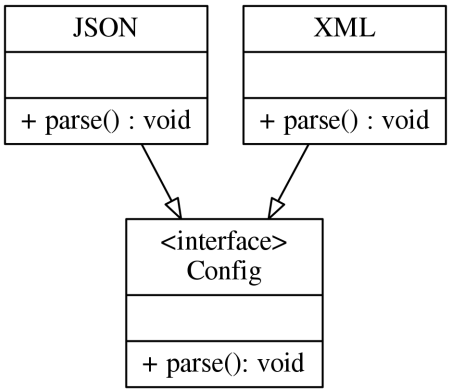
Blok *Zařízení* implementuje čtení dat ze senzorů. Data budou ze zařízení vyčtena na žádost centrálního bloku – při žádosti dostává řídicí blok pouze aktuální dostupnou hodnotu 3.4. Aplikace diplomové práce se zaměřuje pouze na práci s prostorovými senzory, tudíž výstupním formátem načtených dat ze *Zařízení* bude struktura obsahující tři vektory:

```
struct coordinates
{
    vec3d gyro;
    vec3d accel;
    vec3d magnet;
};
```

Vec3d je struktura obsahují souřadnice trojrozměrného vektoru. Omezení implementace pouze na prostorové senzory povede ke zúžení obecnosti implementace, avšak práce s vyčtenými daty bude přehlednější. Navíc bude formát dat stále zachován nehledě na použitý senzor a konkrétní implementaci načítání a parsování dat.

3.4.3 Konfigurace

Základem každé složitější aplikace je možnost její parametrizace, která přidává aplikaci na obecnosti. Jelikož je aplikace jednoduchá a nedisponuje síťovou komunikací, je nejvhodnější ukládat konfiguraci do souboru. Existuje více standardních formátů konfiguračních souborů. Mezi nejpoužívanější formáty v praxi patří (dle autorových zkušeností s mnoha C a C++ knihovnami a aplikacemi) bezesporu INI, XML, JSON, CFG, YAML,...V práci bude použit formát JSON, který je jednoduchý a především také velice dobře čitelný. Navíc je formát přesně specifikovaný a lze jej jednoduše validovat (na rozdíl od INI).



Obrázek 3.2: Rozhraní tříd

3.5 Obecné rozhraní

Během návrhu aplikace je nutné neustále myslet na skutečnost, že výsledný software má být snadno rozšiřitelný a grafické rozhraní obecné. Ke splnění předepsaných požadavků je vhodné rozdělit zdrojové kódy na rozhraní (tzv. „interface“) a implementaci. Interface definuje metody a proměnné, které je možné u dané třídy používat. Konkrétní implementace určuje samotné chování třídy. Při použití obecně napsaného kódu programátor používá pouze rozhraní daných tříd bez toho, aniž by nutně znal konkrétní implementaci (chování metod je popsáné v dokumentaci). Navíc přidáním implementace je možné nadefinovat nové chování třídy při zachování stejného rozhraní – není nutné refaktorovat celou aplikaci. Na obrázku 3.2 lze vidět rozdělení tříd pro případ parsování konfiguračních souborů. Z obrázku vyplývá, že programátor používá pouze metodu *parse()* zadefinovanou v rozhraní *Config*, ale chování použité metody závisí na konkrétní implementaci metody *parse()* ve třídách *JSON* a *XML*.

Rozšířitelnost aplikace je důležitá především pro další vývoj a údržbu aplikace. Aplikaci diplomové práce je možné rozšiřovat naprogramováním specifické implementace k již připraveným rozhraním. Příkladem může být podopora jiného prostorového senzoru, či zvolení jiné grafické knihovny pro 3D vizualizaci, doimplementování dalších vlastností aplikace, podpora více formátů konfiguračního souboru, apod.

Obecné grafické rozhraní představuje skupinu tříd vizualizace, které jsou nevázané na aktuálně používanou grafickou knihovnou 3.3 a vytvářejí rozhraní pro konkrétní implementaci. Pro zachování jednoduchosti implementace bylo s konzultantem práce dohodnuto, že samotná aplikace nebude určovat rozložení a počet ovládacích grafických prvků v uživatelském rozhraní, ale bude pouze určovat jejich obsah a chování. Rozložení uživatelských prvků musí být definováno předem (například v jazyce QML 3.2).

4 Řešení problému

Analýza z kapitoly 2 a návrh z kapitoly 3 dávají jasnou představu, jaký typ hardware a jaký software by měl být použit k vyřešení problému. V kapitole jsou popsány zásadní důvody k výběru konkrétního embedded systému a prostorového senzoru. Dále následující text popisuje problémy s kompilací knihovny Qt 3.3. Výstupem kapitoly je detailní popis přípravné fáze, po které již nastupuje implementace a testování aplikace.

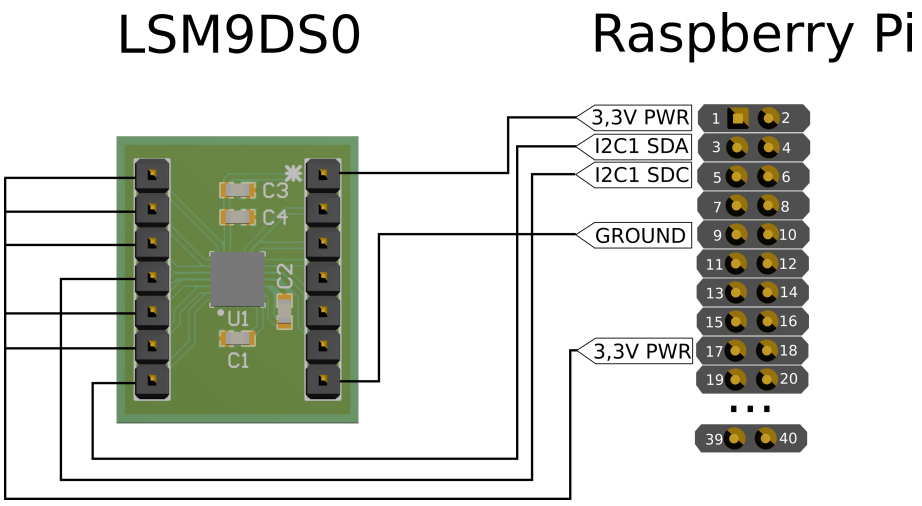
4.1 Prostorový senzor

Prostorový senzor byl vybrán na základě interních čidel. Podle analýzy prostorových senzorů 2.2 je zřejmé, že k dispozici byly dvě konkrétní řešení – GiTy a LSM9DS0. Zásadním problémem ukazovátka GiTy je chybějící čidlo magnetometru. Defekt gyroskopu, tzv. „Gyroskop drift“ 2.3.2 poté není možné kompenzovat při otáčení okolo osy Z. Ač je GiTy již hotovým řešením, kvůli zmíněnému defektu je vhodné použít druhé zařízení. Navíc, jelikož je LSM9DS0 pouze senzorem nikoli celým řešením, je možné v dalším vývoji navrhnout vlastní ergonomické pouzdro a bezdrátovou konektivitu v závilosti na budoucím použití v odvětví rehabilitace.

Senzor LSM9DS0 se připojuje přes I^2C a pro jeho používání je nutné doinstalovat modul (ovladač) do jádra linuxového operačního systému. Modul se kompiluje ze zdrojových souborů. Původně byly ovladače nastaveny tak, aby zařízení posílalo data každých 100 ms (tzv. „poll_interval“). Pro plynulou vizualizaci je potřeba překreslovat minimálně 30 snímků za sekundu, což odpovídá minimálně třikrát rychlejšímu odesílání dat. Po dohodě s vedoucím práce byl „poll_interval“ před kompilací nastaven na hodnotu 20 ms.

4.2 Embedded systém

Volba konkrétního embedded systému byla určena na základě testování. Nejprve byl k dispozici minipočítač Acqua A5 2.1.1. Během práce s deskou se zjistilo, že neobsahuje grafický čip. Vykreslení 3D vizualizace je ovšem možné i bez grafického čipu pouze za podpory procesoru a využití alternativních knihoven k OpenGL (například Mesa3D). K ověření, zda je výkon procesoru dostatečný pro složitější vizualizaci, byla zkompilována knihovna Qt pro desku Acqua A5. Pro úspěšnou kompilaci bylo nutné doinstalovat mnoho knihoven potřebných k základnímu použití – jelikož na desce byl



Obrázek 4.1: Schéma připojení LSM9DS0 k Raspberry Pi

velice omezený linuxový systém a knihovny v software repozitářích zcela chyběly, nebo byly zastaralé, bylo nutné závislosti zkompilovat ze zdrojových souborů. Při testování knihovny Qt na zařízení Acqua se ověřilo, že i bez grafického čipu je možné plynule vykreslit jednoduchou vizualizaci ve 2D, ovšem při použití 3D či složitější 2D vizualizace začal být problém s výkonem a překreslování dosahovalo rozmezí 5-15 FPS, což je pro běžné používání nepřijatelné.

Po předchozím zjištění se naskytla možnost vyzkoušet minipočítač Raspberry Pi B+ 2.1.2. Alternativa již obsahuje grafický čip a vykreslení 3D vizualizace bylo bez obtíží. Navíc instalace Qt zabrala řádově méně času než instalace na původně použitou desku, jelikož Raspberry je rozšířenějším minipočítačem a existuje k němu velká podpora (obsáhlejší softwarový repozitář, diskuzní fóra, oficiální dokumentace,…). Minipočítač byl ovšem pouze zapůjčen od studenta. Aby bylo možné pracovat s vlastním minipočítačem, vedoucí práce zajistil zařízení Raspberry Pi 3 2.1.3, které bylo nakonec použito pro implementaci a testování aplikace.

I přes skutečnost, že Raspberry Pi 3 poskytuje dostatečný výkon i pro správce oken, je z hlediska přenositelnosti aplikace vhodné používat grafickou platformu 2.4 nezávislou na správci oken. Tím je možné ušetřit cenné prostředky na skutečném vestavěném systému. V diplomové práci je použita knihovna Qt 3.3, která umožňuje přepínání závilosti na konkrétní platformě při spouštění aplikací – jednu aplikaci je možné multiplaformě spustit na jakémkoli stroji. Během práce na praktické části bude použita platforma *EGLFS*, jelikož je doporučená pro embedded systémy a poskytuje nejlepší poměr mezi funkčností a využitými prostředky.

4.3 Připojení senzoru

Vybraný senzor LSM9DS0 2.2.2 se připojuje přes sběrnici I^2C. Pro snadné připojení vedoucí diplomové práce vytvořil speciální redukci. Schéma zapojení prostorového senzoru k vybranému minipočítači je znázorněno na obrázku 4.1.

4.4 Kompilace Qt

Jak již bylo popsáno, Qt 3.3 je velice komplexní a modulární knihovna. Chování a multiplatformnost knihovny se liší v závislosti na nastavení kompilace. Nastavením je možné ovlivnit používanou grafickou platformu 2.4, přidat či odebrat kompilované moduly, nastavit kompilátor křížové kompilace, cestu k „sysroot“ 3.1.1 a další důležité cesty, které se při kompilaci používají. Na dostupném Raspberry byla knihovna zkompilována s následující konfigurací:

```
./configure -opengl es2 -make libs -make tools
            -sysroot /home/lukas/raspberry/sysroot
            -device linux-rasp-pi2-g++ -device-option
             CROSS_COMPILE=arm-linux-gnueabihf-g++
            -prefix /usr/local/rasp -opensource
            -nomake examples -nomake tests -confirm-license
            -skip qtwebengine -skip qtsensors -skip qtgamepad
            -skip qtmacextras -skip qtandroidextras
            -prefix/usr/local/qt5pi
            -extprefix /home/lukas/raspi/qt5pi
            -hostprefix /home/lukas/raspi/qt5
```

Z nastavení lze vidět, že kompilace je specifikovaná přímo pro zařízení Raspberry Pi. Zároveň je vhodné, aby knihovna v rámci ušetření času byla zkompilována křížově 3.1.1 na výkonném stroji. Například na čtyřjádrovém procesoru s frekvencí 2,2 GHz se knihovna s využitím všech jader kompiluje více jak 30 minut.

Výstupem konfiguračního skriptu knihovny je seznam závislostí na ostatních knihovnách. Pro správnou kompilaci knihovny není nutné splnit všechny závilosti, avšak knihovna poté nemusí pracovat správně. Závislosti Qt lze doinstalovat z oficiálních softwarových repozitářů nebo zkompilovat ze zdrojových souborů.

5 Implementace

Po provedené analýze 2, návrhu 3 a dořešení všech dalších vzniklých problémů 4 nastává proces implementace. Jak bylo popsáno v návrhu, aplikace je rozdělena na jednotlivé části 3.4. Každá část je nezávislá, tudíž jsou vytvořeny samostatné funkční bloky, které je možné portovat i do jiných aplikacích. Výsledek bude poté spojen jednou samostatnou třídou, která dle návrhu tvoří funkční blok *Aplikace*.

Jelikož má výsledný prototyp pouze testovací charakter, není nutné se dopodrobna zabývat návrhem grafického rozhraní (rozložením jednotlivých prvků, designem,...). Uživatelské rozhraní je proto jednoduché a obsahuje pouze základní ovládací prvky potřebné pro ovládání vizualizace. Do budoucna je ovšem možné rozhraní a ovládací prvky dále jednoduše rozšiřovat. Výsledná aplikace je zobrazena na obrázku 5.1. Vizualizační část obsahuje 3D krychli, jejíž orientace je vázaná na data z prostorového senzoru. Druhou částí uživatelského rozhraní jsou jednoduché ovládací prvky, které ovlivňují vizualizaci a zpracování dat.

Na některých embedded operačních systémech mohou být problémy s programovými výjimkami, proto jsou všechny zdrojové kódy napsané jako „exception safe“, neboli žádný ze zdrojů nevyvolává výjimky a tudíž je není nutné ošetřovat. Chyby jsou ošetřeny pomocí standardních nástrojů C++ knihovny.

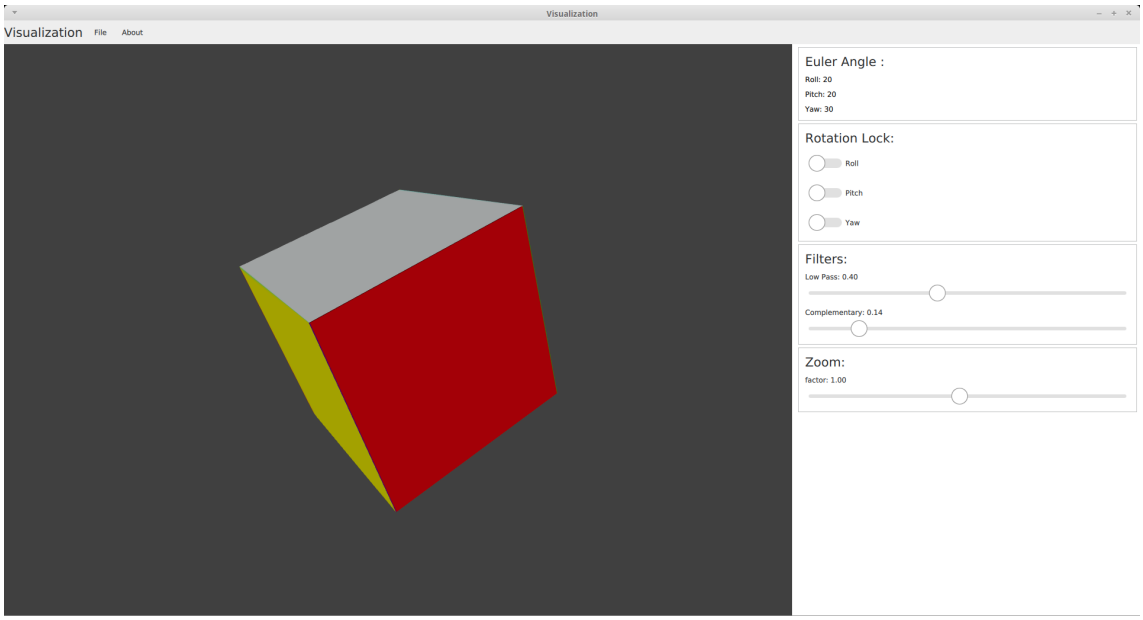
V kapitole je nejdříve popsána struktura aplikace a kompilace, dále způsob dokumentace zdrojových kódu a poté samotná implementace jednotlivých částí kódu. Detailněji popsána jsou pouze veřejná rozhraní zdrojových kódů. Privátní metody, které zpřehledňují samotnou implementaci chování, jsou vysvětlené v přiložené HTML dokumentaci.

5.1 Struktura a kompilace

Zdrojové kódy jsou rozděleny do jednotlivých částí 3.4. Každá část představuje jeden samostatný funkční blok. Části jsou od sebe odděleny pomocí namespace[1]. Zdrojové kódy pro každý funkční blok navíc obsahují namespace „detail“, který obaluje interní (neveřejné) implementace. Pro uživatele knihovny jsou poté zpřístupněny pouze důležité části zdrojových kódů.

Knihovna Qt 3.3 využívá k nastavení kompilace vlastní soubory, jejichž struktura je jednodušší než klasicky používaný makefile. Qt projektové soubory mají příponu „.pro“. Pomocí nástroje *qmake* (nástroj, vznikl při sestavení knihovny Qt;

[1] Jmený prostor – slučuje zdrojové kódy řešící stejný problém do jedné skupiny



Obrázek 5.1: Implementovaná aplikace

je specifikován pro konkrétní architekturu) je poté ze souboru „.pro“ vytvořen klasický makefile soubor, pomocí kterého se projekt kompiluje. Pokud se překládá pro jinou architekturu, je nutné používat *qmake* a kompilátor cílové architektury.

5.2 Dokumentace

Pro udržitelnost aplikace a případný další rozvoj je důležité dobře zdokumentovat napsané zdrojové kódy. Mezi standardní nástroje generování dokumentace patří obecně známý a textově orientovaný Doxygen [39]. Nástroj je zvolen na základě předchozí dlouholeté zkušenosti autora.

Doxygen generuje dokumentaci ve formátu HTML nebo LaTeX. Uživatelské rozhraní vygenerované dokumentace může působit poněkud zastarale, avšak je přehledné a zobrazuje všechny důležité informace. Dokumentace zdrojových kódů má formu anotacích (strukturovaných komentářů). Existují dva možné formáty anotací, které umí nástroj doxygen zpracovat – vlastní doxygen formát a *javadoc*. V diplomové práci bude použit formát dokumentace *javadoc*, jelikož se jedná o velice rozšířený formát dokumentace zdrojových kódů napříč všemi dokumentačními nástroji a programovacími jazyky.

Ovládacím souborem je tzv. „Doxyfile“, ve kterém je možné nastavit výstup vygenerované dokumentace. V práci je již řídicí soubor obsažen a není nutné jej nastavovat. Nástroj je nastaven tak, aby výstupní dokumentace byla uložena do složky „doc/“ ve formátu HTML. Zároveň projekt obsahuje soubor „README.mk“, což je soubor napsaný ve formátu *markdown* a tvoří úvodní stranu vygenerované dokumentace.

5.3 Licence

Implementace je postavena na standardních C++ knihovnách, knihovně Qt 3.3 a knihovně JSON pro zpracování stejnojmenného formátu souboru. Standardní knihovny jsou licencovány pod GPL licencí – je možné provádět jakékoli úpravy, modifikace a dále rozšiřovat zdrojový kód v komerčním i nekomerčním využití. Knihovna Qt má rozdělené licence pro komerční a nekomerční použití. V diplomové práci bude pro Qt použita použita nekomerční open-source verze licence LGPLv3 – je nutné poskytnout zdrojový kód výsledné aplikace, apod. Dále je nutné upozornit, že licence LGPLv3 se nevztahuje na všechny moduly knihovny Qt (některé jsou stále licencované jako proprietární software). V poslední řadě knihovna JSON je vydána pod licencí MIT – bez omezení je možné zdrojový kód používat, kopírovat, modifikovat, apod. Detailnější popis je dostupný v [40].

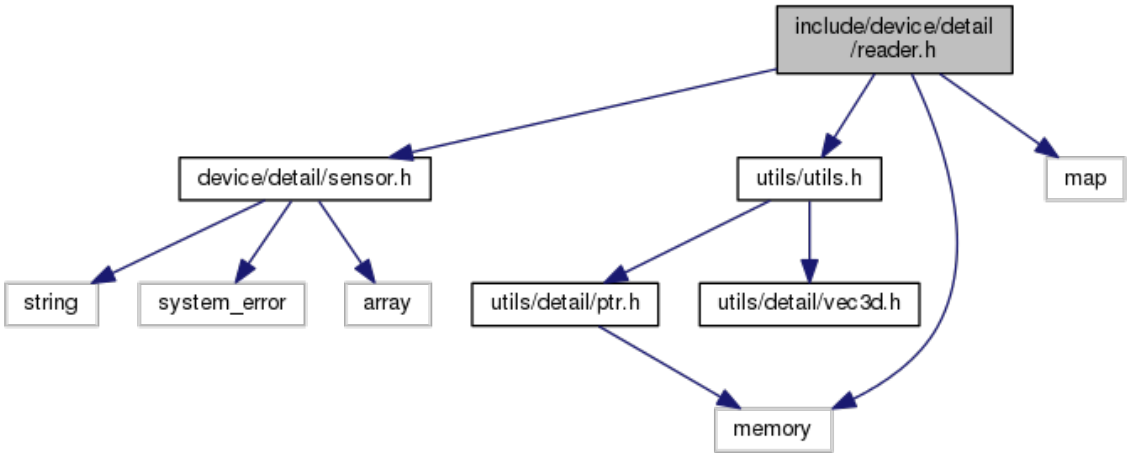
Jelikož je knihovna Qt licencována pomocí LGPLv3 je nutné, aby všechny projekty odvozené od zmíněné knihovny byly licencovány stejnou licencí. Zároveň je použita knihovna JSON s MIT licencí, na kterou se nevztahuje podmínka dědičnosti. Z uvedených faktů vyplývá, že výsledný zdrojový kód aplikace je vydán pod licencí LGPLv3.

5.4 Pomocné struktury

V aplikaci bylo nutné vytvořit pouze jednu pomocnou strukturu *vec3d*. Struktura reprezentuje trojrozměrný vektor čísel a je dostupná napříč všemi funkčními bloky. Chování struktury je navíc rozšířeno veřejnými metodami, které umožňují provádět s vektory základní matematické operace (normalizace, výpočet velikosti,...). 3D vektory se používají ve vizualizaci pro určení pozice a případně směru pohybu. V diplomové práci je struktura použita také pro reprezentaci načtených dat ze zařízení 3.4.2. Rozhraní struktury je dáno metodami:

- *vec3d()* představuje konstruktor. Jestliže nejsou zadány argumenty konstruktoru, je vytvořena prázdná instance s nulovými souřadnicemi. Pokud jsou argumenty zadány, je vektor nastaven na konkrétní hodnoty. Struktura má také zadefinovaný kopírovací konstruktor, který pouze vytvoří kopii z jiného již existujícího vektoru.
- *normalize()* metoda normalizuje vektor, tzn. přepočítá souřadnice vektoru tak, aby vektor měl jednotkovou délku.
- *size()* navrací vypočítanou velikost vektoru.

Třída *vec3d* dále implementuje rozhraní pro matematické operace s vektorem. Vektor je možné násobit (resp. dělit) skalárem, dále je možné dva vektory sečíst či odečíst. Rozhraní matematických operací zjednodušuje práci s vektory a zpřehledňuje zdrojový kód.



Obrázek 5.2: Diagram závislostí pro blok *Zařízení*

5.5 Zařízení

Prvním implementovaným funkčním blokem 3.4 je *Zařízení*. Blok (knihovna) pro zařízení umožňuje číst data z prostorových senzorů. Pro správnou funkčnost prostorového senzoru může být nutná jeho počáteční inicializace (zavedení modulů do jádra systému, Bluetooth spárování,apod.). Funkční blok předpokládá, že zmíněná inicializace byla provedena již na úrovni operačního systému (tzn. nemá připravené rozhraní pro zavedení zařízení do systému). Vytvořená knihovna je postavena pouze na standardních C++ knihovnách a poskytuje vhodné rozhraní pro implementaci konkrétních prostorových senzorů.

Reader třída je základem celé implementace pro čtení dat ze senzorů. Senzory jsou do třídy zaregistrovány pod zadaným jménem a následně jsou vyčtena data ze všech senzorů najednou. Například pokud máme k dispozici prostorový senzor s čidly akcelerometr, magnetometr a gyroskop, je navrácena struktura 3.4.2 obsahující aktuální data ze všech senzorů. Tento přístup zajišťuje obecnost řešení – data mohou být čtena z jednoho či více čidel. Na obrázku 5.2 lze vidět diagram závislotí třídy reader. Rozhraní třídy je definováno následovně:

- *bind()* zaregistruje senzor do třídy reader pod zadaným konkrétním názvem. Jestliže se registrace nepovede (senzor se stejným názvem již existuje) je navrácena standardní C++ chybová struktura.
- *unbind()* zruší registraci zaregistrovaného senzoru dle zadaného názvu. Jestliže senzor s daným názvem zaregistrován nebyl, je navrácena chyba.
- *read()* vyčte data ze všech zaregistrovaných senzorů a navrátí vhodnou strukturu 3.4.2. Pokud dojde k chybě vyčtení dat (například při náhlém odpojení senzorů), je navrácena chyba. Metoda čte vždy jen jeden aktuální záznam.

Senzor představuje prostorový senzor, nebo čidlo prostorového senzoru (pokud je senzor složený z více čidel). Všechna zařízení jsou v Linuxu dostupná jako soubory, do kterých se ukládají výstupy. Ze souborů zařízení je možné číst vždy aktuální (poslední) hodnotu načtenou ze zařízení. Třídě je nutné tedy předat cestu k souboru, ze kterého bude data vyčítat. Navrácená data jsou v podobě vektoru, který obsahuje data vyčtená ze senzoru/čidla:

$$(gyro_x, gyro_y, gyro_z, accel_x, accel_y, accel_z, magnet_x, magnet_y, magnet_z) \tag{5.1}$$

Jestliže senzor například obsahuje pouze čidlo gyroskopu, vyplní se vektor na daných indexech a ostatní souřadnice zůstanou prázdné (nulové). Z navrácených vektorů třída *reader* vytvoří strukturu vhodnou pro další zpracování. Třída je abstraktní, vytváří pouze rozhraní pro následné implementace konkrétních senzorů. Veřejné rozhraní třídy je tvořeno metodami:

- *open()* na základě předané cesty k senzoru připraví třídu ke čtení dat. Pokud není možné otevřít daný soubor, je třída nastavena do chybného stavu.
- *is_open()* otestuje, zda je soubor správně otevřen ke čtení.
- *close()* zavře čtený soubor. Metoda je volána i v destruktoru třídy, aby se nezapomínalo na uzavření otevřeného souboru.
- *fail()* otestuje zda je třída v chybovém stavu. Metoda *is_good()* poté navrací negaci navraceného stavu metodou *fail()*.
- *operator bool()* poskytuje příjemnější rozhraní pro testování stavu třídy.
- *read()* čte data z otevřeného souboru. Pokud dojde k chybě (konec souboru), je navrácen chybový kód. Data jsou navrácena ve formátu vektoru 5.1.

5.5.1 LSM9DS0 implementace

Jelikož každý prostorový senzor poskytuje jiný formát dat, je nutné implementovat každý jeden senzor specifickým způsobem. Třída *senzor* je abstraktní a vytváří rozhraní pro prostorové senzory či jednotlivá čidla.

LSM9DS0 senzor je složen ze tří čidel. Každé čidlo se připojuje k operačnímu systému jako samostatné zařízení (ve filesystému jsou tři dostupné soubory, ze kterých je možné data číst). Základem implementace je třída *lsm9ds0*, která implementuje rozhraní třídy *senzor*. Implementace správně připraví instanci třídy pro čtení z konkrétního souboru a parsuje data pomocí modulů jádra operačního systému do vektoru 5.1.

Třída *lsm9ds0* je stejně jako *senzor* pouze abstraktní třídou. Konkrétní potomci třídy *lsm9ds0* musí implementovat metodu *assign_to_motion_vector()* pro přiřazení dat čidla na správné indexy vektoru 5.1. Specifické implementace jsou *lsm9ds0_gyro*, *lsm9ds0_accel* a *lsm9ds0_magnet*.

5.6 Vizualizace

Hlavní částí výsledné aplikace je nepochybně 3D vizualizace a uživatelské rozhraní. K udržení obecnosti řešení a především nezávilosti na grafických knihovnách 3.3 je vytvořeno obecné rozhraní všech tříd. Chování aplikace je poté závislé na specifické implementaci. Neboli pokud dojde k rozhodnutí, že je potřebné vyměnit knihovnu uživatelského rozhraní, nebude nutné přepisovat celý zdrojový kód, avšak pouze se doplní specifická implementace pro danou knihovnu. Zmíněný přístup může být vhodný především ve chvíli, kdy kód pro vizualizaci používá více aplikací – bez větších obtíží je možné změnit závilost na grafické knihovně u všech aplikací najednou.

K rozdělení rozhraní a samotné implementace je využita kompozice, která na rozdíl od dědičnost nedefinuje tak silné vazby mezi jednotlivými třídami. Rozhraní jsou tvořena šablonovými třídami (parametr šablony je typ konkrétní implementace). Kvůli své složitosti zde není uveden diagram závislostí, lze jej však najít v přiložené dokumentaci.

Gui tvoří základní rozhraní pro správu uživatelského rozhraní. Z hlediska charakteru aplikace se předpokládá, že bude vždy zobrazeno pouze jedno hlavní okno (vhodné pro vykreslování přímo do framebufferu 2.4). Třída poskytuje rozhraní pro otevření/zavření uživatelského rozhraní a dále dává možnost zaregistrovat zpětná volání na důležité události. U grafických aplikací je nutné, aby kód grafického rozhraní byl spuštěn v hlavním vlákně. Jelikož je spuštění rozhraní blokující, umožňuje třída *gui* nastavit funkci, která bude spuštěna v pracovním vlákně (nutné pro asynchronní zásahy do uživatelského rozhraní). Veřejné rozhraní třídy obsahuje metody:

- *gui()* konstruktor, který inicializuje třídu do výchozí stavu.
- *open()* otevře grafické rozhraní. Po otevření zavolá zpětné volání, které bylo nastaveno metodou *on_opened()* a dále v pracovním vlákně spustí funkci, která byla nastavena pomocí *async_handler()*.
- *is_opened()* otestuje, zda je uživatelské rozhraní spuštěno.
- *close()* zavře vizualizaci a ukončí pracovní vlákno.
- *on_opened()* nastavení tzv. „callbacku“ [2], který se zavolá po úspěšném otevření uživatelského rozhraní.
- *on_close()* podobně jako přechozí metoda nastaví callback, který se volá po uzavření vizualizace.
- *async_handler()* nastaví funkci, která se zavolá v pracovním vlákně. Funkce se nevolá opakovaně. Periodicitu funkce je nutné zařídit v těle předaného callbacku.

- *is_good()* testuje, zda je třída správně inicializovaná a uživatelské rozhraní je zobrazeno.
- *operator bool()* poskytuje příjemnější rozhraní pro testování stavu aplikace.

Všechny nastavené callbacky mají v argumentu nastavenou referenci na třídu *window*. U funkce, která se volá v pracovním vlákně je ovšem nutné zajistit mezivláknovou bezpečnost při zásahu do vizualizace. Aktualizace uživatelského rozhraní probíhá z pracovního, nikoli hlavního, vlákna a může docházet ke konfliktům při zápisu do paměti. Proto je vhodné, aby dle návrhu 3.4.1 byly všechny vizualizační třídy napsány jako „thread safe“.

Window tvoří aplikačně specifické, avšak z hlediska implementace stále obecné rozhraní pro jedno konkrétní okno. Drží instance tříd, které se starají o zpracování a vykreslení jednotlivých částí okna (*single_model_canvas*, *controls* a *spinner*). Zmíněné instance jsou neinicializované, tzn. že po vytvoření instance *window* je nutné ještě inicializovat všechny její části. Třída navíc umožňuje nastavit rozměry a titulek okna. Metody poskytované třídou jsou:

- *window()* konstruktor třídy, který své argumenty předává do metody *init()*. Argumenty jsou rozměry okna a titulek a mají nastavené takové výchozí hodnoty, že šířka okna je 1024 pixelů, výška 768 pixelů a titulek okna je „Application title“.
- *init()* inicializuje nové okno na základě předaných argumentů (výchozí hodnoty jsou stejné jako u předchozí metody). Dále vytvoří instance tříd, které se starají o zpracování jednotlivých částí oken.
- *width(), height() a title()* jsou metody, které navracejí vlastnosti okna (šířku, výšku a titulek). Metody je možné volat i s argumenty, které nastavují dané vlastnosti okna a informují uživatelské rozhraní o změnách – dojde k překreslení vizualizace s nově nastavenými vlastnostmi.
- *on_resize()* nastaví zpětné volání, které se zavolá v případě, že uživatel změní rozměry okna.
- *on_exit()* umožňuje předat callback, který se aplikuje při uzavření okna nebo při výběru položky „Exit“ z menu aplikace.
- *on_save()* poskytuje rozhraní pro nastavení callbacku, který se zavolá při výběru položky „Save“ z menu aplikace.
- *canvas(), controls() a spinner()* jsou metody, které navrací referenci na instance, které ovládají jednotlivé části okna. Reference je možné využít pro inicializaci a rutinní práce se třídami.

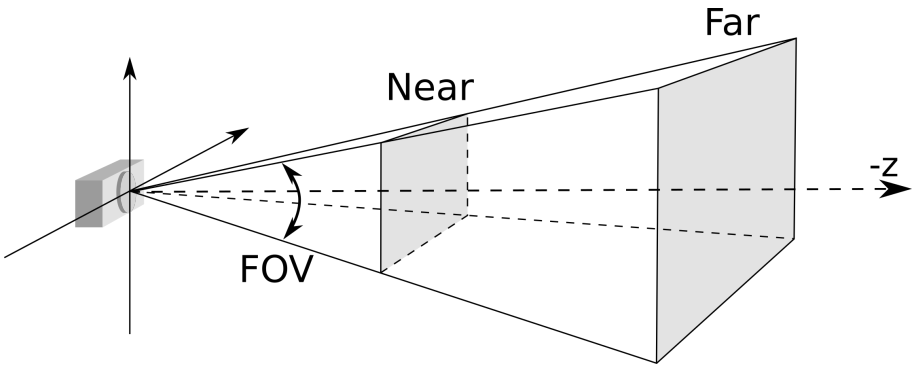
[2]Zpětné volání - zaregistrovaná funkce se zavolá synchronně či asynchronně až ve chvíli, kdy je to potřebné.

Single_model_canvas představuje část uživatelského rozhraní, která vykresluje 3D vizualizaci. Z hlediska charakteru aplikace je třída připravena pouze pro práci s jedním 3D modelem. U vizualizace je možné nastavit barvu pozadí a ambientního světla. Dále drží třída *single_model_canvas* instance tříd *model* a *camera*. Instance jsou neinicilizované; pro správnou funkčnost 3D vizualizace je nutné třídy před použitím nastavit do výchozího stavu. Veřejné rozhraní je následující:

- *single_model_canvas()* představuje konstruktor třídy. Je možné jej volat bez argumentů a nebo jako argumenty nastavit barvu pozadí a barvu ambientního světla. Výchozí hodnota pro barvu pozadí je 0x404040 a pro barvu světla 0xffffff. Barvy jsou uloženy jako neznaménková celá čísla, avšak z důvodu přehlednosti je vhodné je zadávat v šestnáctkové soustavě – každé dva bajty určují vliv jedné ze složek RGB. Argumenty jsou předány do metody *init()*.
- *init()* inicializuje třídu do výchozího funkčního stavu. Podobně jako u předchozí funkce je možné metodě předat argumenty barvy pozadí a ambientního světla.
- *background_color() a light_color()* navracejí nastavené barvy. V případě, že jsou metody zavolány s argumentem konkrétní barvy je daná vlastnost nastavena ve třídě a vizualizace se informuje o vzniklé změně – dojde k překreslení.
- *model() a camera()* jsou metody, které navracejí referenci na dané třídy. Pomocí reference je možné třídy inicializovat do počátečního stavu a následně menit jejich vlastnosti.
- *is_good()* testuje, zda při inicializaci 3D vizualizace nedošlo k chybě.
- *operator bool()* poskytuje lepší rozhraní pro testování chyby metodou *is_good()*.

Model3d reprezentuje 3D model, který je v uživatelském rozhraní vykreslován v trojrozměrné vizualizaci (*single_model_canvas*). Třída předpokládá, že model bude načten ze souboru, tudíž při své inicializaci očekává cestu k danému modelu. Informace o materiálu a textuře 3D objektu již musí být uložené v načítaném souboru. Dále třída poskytuje rozhraní, pomocí kterého je možné s modelem rotovat. Střed 3D objektu je nastaven do bodu $[0, 0, 0]$ a rotace se provádí vzhledem ke stejnému bodu. Třída poskytuje následující veřejné rozhraní:

- *model3d()* je konstruktorem třídy. Může být zavolán s prázdnými argumenty, poté je ovšem nutné samostatně inicializovat instanci, nebo je zavolán s argumentem cesty k načtenému souboru a dojde k načtení modelu pomocí funkce *load()*.
- *load()* dle zadané cesty k souboru načte 3D model. Jestliže se načtení souboru nepovede, je třída nastavena do chybového stavu.
- *file_path()* navrací cestu, ze které byl model načten.

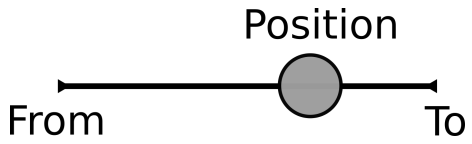


Obrázek 5.3: Perspektivní projekce

- *pitch(), roll() a yaw()* jsou metody, které navrací rotaci modelu vzhledem ke své počáteční pozici (absolutní rotace) v podobě Eulerových úhlů 2.2. Metody je možné zavolat i s argumentem konkrétního úhlu (ve stupních) a nastavit tak rotaci okolo dané osy.
- *rotation()* navrátí *vec3d* Eulerových úhlů v pořadí *roll*, *yaw*, *pitch*. Pořadí úhlů vychází ze souřadnicového systému trojrozměrné vizualizace, kde je zvykem mít pravotočivý systém a navíc svislou osou je osa Y (nikoli Z, jak bývá zvykem například v axonometrii). Pokud je do metody předán *vec3d* daných úhlů, je vizualizace informována o překreslení, resp. otočení, modelu.
- *is_good()* otestuje, zda byl 3D model správně načten ze souboru.
- *operator bool()* poskytuje příjemnější rozhraní pro testování stavu třídy.

Camera je třída, která poskytuje rozhraní pro nastavení vlastností kamery v trojrozměrné vizualizaci. Kamera určuje, jak budou uživateli zobrazeny objekty, které jsou ve vizualizaci umístěny. Kromě nastavení polohy kamery a bodu zájmu (směr záběru kamery) jsou k dispozici také další nastavení umožňující měnit samotnou perspektivní projekci (obrázek 5.3). Veřejné rozhraní obsahuje metody:

- *camera()* je prázdným konstruktorem třídy, který nastaví pozici kamery do bodu $[0, 0, 5]$ a bod zájmu do středu projekce. Konstruktor také nastavuje výchozí perpektivní projekci, kde úhel FOV je 60°, a snímaná oblast je nastavena v intervalu $\langle 0,1; 1000 \rangle$.
- *look_at()* navrací bod zájmu (bod, na který kamera směřuje). Jestliže je metoda zavolána s argumentem *vec3d*, poté aktualizuje bod zájmu a informuje vizualizaci o vzniklé změně.
- *position()* metoda vrátí aktuální pozici kamery. Podobně jako u předchozí metody je možné ji předat argument *vec3d* pro nastavení pozice.
- *fov(), near() a far()* jsou metody, které navracejí vlastnosti perspektivní projekce podle obrázku 5.3. Metodám je možné předat argument, který nastaví



Obrázek 5.4: Ovládací prvek *range*

danou vlastnost a informuje vizualizaci o změně, kde *fov()* se nastavuje ve stupních a jednotky *near()* a *far()* závisí na konkrétní zobrazované scéně a charakteru aplikace.

- *zoom()* poskytuje rozhraní pro získání a nastavení faktoru přiblížení.

Controls představuje třídu, která pouze drží kolekci ovládacích prvků v aplikaci. Podobně jako *window* je třída *controls* aplikačně specifická, avšak stále vytváří obecné aplikační rozhraní. Všechny ovládací prvky jsou navráceny v podobě referencí na jednotlivé třídy, které poskytují rozhraní pro práci s danými typy ovládacích prvků. Držené instance nejsou inicializovány a jejich počáteční nastavení je nutné provést před spuštěním uživatelského rozhraní. Metody tvořící veřejné rozhraní třídy *controls* jsou:

- *zoom(), complementary() a low_pass()* jsou metody, které vracejí reference na ovládací prvky popsané třídou *range*. Prvek *zoom()* nastavuje faktor přiblížení kamery ve 3D vizualizaci. Dále reference získané pomocí *complementary()* a *low_pass* umožňují měnit nastavení stejnojmenných filtrů 2.3.

- *roll_lock(), pitch_lock() a yaw_lock()* jsou metody, které navracejí reference na třídy *toggle*. Ovládacími prvky lze ovlivnit, podle kterých os je možné otáčet 3D model ve vizualizaci (neboli rotaci lze v jedné či více osách uzamknout).

Range třída reprezentuje ovládací prvek, kterým lze nastavit hodnotu z daného číselného intervalu $\langle from, to \rangle$ (obrázek 5.4). Rozhraní třídy pouze umožňuje nastavit zpětné volání, které se aplikuje při výběru nové pozice z intervalu. Samotné nastavení ovládacího prvku musí být provedeno při inicializaci a dále jej není možné měnit. Veřejné rozhraní třídy je následující:

- *range()* je konstruktorem třídy. Pokud je zavolán bez argumentů, jsou použity výchozí hodnoty pro nastavení ovládacího prvku. Přednastavené hodnoty jsou: from = 0, to = 1, default_value = 0,5 (výchozí pozice) a step = 0 (krok změny; jeli nastaven na 0, poté je krok změny nekonečně malý). Konstruktor předává argumenty do metody *init()*.

- *init()* inicializuje třídu do výchozí stavu. Neboli nastaví ovládací prvek pomocí předaných argumentů.

- *from(), to(), step() a default_value()* jsou metody, které umožňují zpětně číst nastavení třídy.

- *position()* vrací aktuální vybranou hodnotu v intervalu $\langle from, to \rangle$. Jestliže je nastaven krok změny na jinou hodnotu než nula, poté je navrácená pozice celočíselným násobkem zadaného kroku.

- *on_changed()* nastaví callback, který se zavolá ve chvíli, kdy je aktuální hodnota (pozice) ve třídě *range* změněna.

Toggle třída vytváří rozhraní pro ovládání dvoustavového přepínače či jiného prvku, jehož vlastnosti lze popsat pouze dvěma stavy – příkladem může být zaškrtávací tlačítko (tzv. „checkbox“). Stavy jsou reprezentovány booleovskými hodnotami „true“ (zapnuto) a „false“ (vypnuto). Rozhraní třídy je jednoduché a umožňuje pouze zjistit a nastavit stav přepínače a zaregistrovat zpětné volání, které se vyvolá při změně stavu provedené z uživatelského prostředí:

- *toggle()* představuje konstruktor třídy, který nastaví přepínač do vypnutého stavu.

- *checked()* je metoda, kterou je možné navrátit aktuální stav přepínače. Jako argument metody je možné přidat booleovskou hodnotu určující nový stav přepínače. Po nastavení nové hodnoty se provede změna v uživatelském rozhraní.

- *on_checked()* umožňuje nastavit callback, který se zavolá ve chvíli, kdy uživatel změní stav přepínače.

Spinner je třída představující tzv. „busy indicator“. Neboli třída poskytuje rozhraní pro ovládání indikátoru, který uživatele informuje o zaneprázdněnosti aplikace (jestliže aplikace provádí déletrvající operaci je nutné, aby uživatel o dané akci věděl). *Spinner* umožňuje aktivovat či deaktivovat indikaci a navíc je možné nastavit zprávu popisující aktuální stav činnosti aplikace. Veřejné rozhraní obsahuje metody:

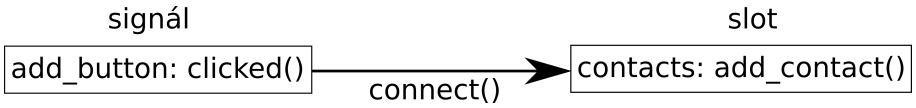
- *spinner()* metoda je konstruktor, který vytvoří prázdnou instanci. Ve výchozím stavu je indikace vypnuta a zpráva indikátoru je prázdná.

- *enable()* navrací, zda je indikátor aktivovaný. Argumentem metody může být booleovská hodnota, která umožňuje inidikátor aktivovat či deaktivovat.

- *label()* je metodou vracející aktuální zprávu *spinneru*. Podobně jako u předchozí metody je možné předat argument, který nastaví zprávu indikace a metoda dále informuje uživatelské rozhraní o změně.

5.6.1 Qt implementace

V diplomové práci je použita knihovna Qt s modulem Canvas3D 3.3. Samotné uživatelské rozhraní lze v Qt napsat pomocí jazyků QML a Javascript 3.2, ve kterých je možné definovat i logiku celé aplikace. Avšak pro větší obecnost aplikace a zejména pro úplné oddělení logiky aplikace od uživatelského prostředí, jsou jazyky QML a Javascript v diplomové práci použity pouze pro vykreslení jednotlivých grafických



Obrázek 5.5: Systém signálů a slotů

prvků. Chování aplikace je poté napsáno v jazyce C++. Lze říci, že funkční blok *Vizualizace* 3.4 je rozdělen na C++ a QML část.

V jazyce C++ je implementováno rozhraní tříd bloku *Vizualizace*, které jsou popsány v předchozím textu. Pro komunikaci se samotným QML uživatelským rozhraním se používá systém signálů a slotů. Zjednoduše lze systém přirovnat k návrhovému vzoru „předplatitel“ [3]. Vyslaný signál je zaregistrován ve frontě signálů knihovny Qt. Jakmile dojde ke zpracování konkrétního signálu, zavolá se funkce, která byla zaregistrovaná jako slot daného signálu (tzn. že před vysláním signálu je nutné propojit signál se slotem) a dojde k obsluze. Systém je znázorněn na obrázku 5.5.

Qt navíc poskytuje dvě makra, která zjednodušují komunikaci C++ kódu s QML. Makro *Q_PROPERTY* zpřístupní pod zadaným názvem konkrétní C++ členskou proměnnou do QML. Aby bylo možné proměnnou používat, je nutné v C++ zadefinovat metody pro získání a nastavení hodnoty dané proměnné. V případě že se nejedná o konstantní proměnnou, je nutné dále dodefinovat signál, který informuje uživatelské rozhraní o změně proměnné (slot signálu je již automaticky zaregistrován při kompilaci QML). Druhé makro *Q_INVOKABLE* umožňuje volat C++ funkci z QML. Přímé zavolání funkce je vhodné pro zpracování uživatelských událostí (stisknutí tlačítka, změna stavu přepínače,...). Implementace rozhraní tříd vizualizace 5.6 pracuje se zmíněnými makry a poskytuje tak plnou kontrolu nad uživatelským rozhraním pouze z C++ kódu.

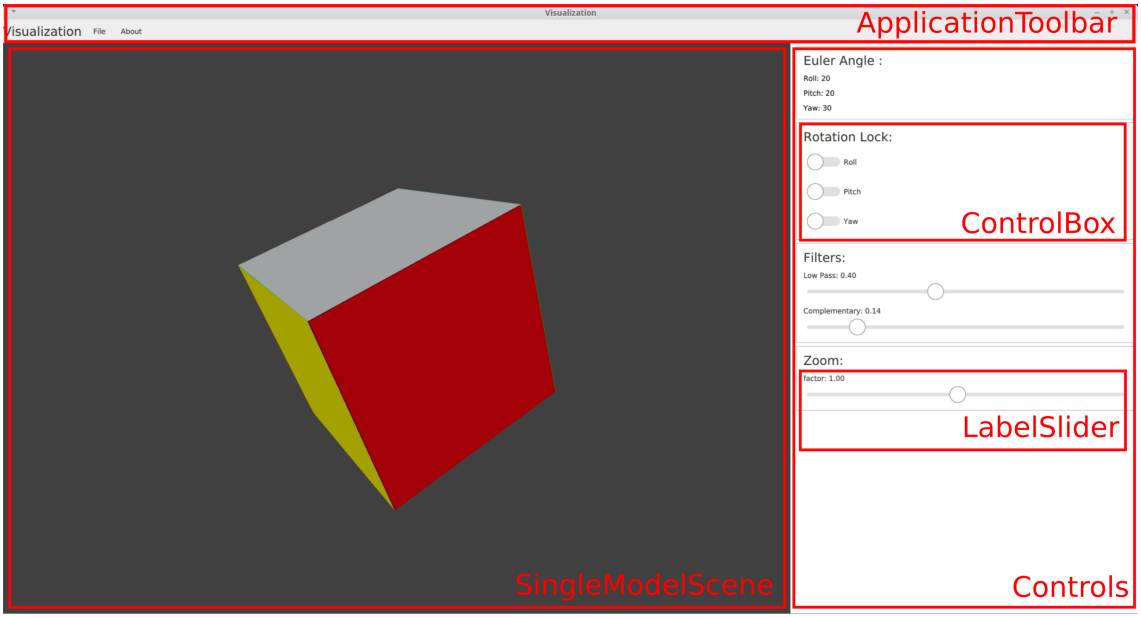
QML komponenty

Zdrojový kód QML je rozdělen na komponenty, které se starají o zobrazení jednotlivých částí vizualizace. Každá komponenta je napsána v odděleném souboru, aby bylo možné komponentu jednoduše použít i v jiném projektu. Na obrázku 5.6 jsou znázorněny jednotlivé komponenty grafického rozhraní. Nestandardní a důležité komponenty jsou popsány v následujícím textu.

ApplicationWindow je komponenta z knihovny Qt a tvoří základ celého zobrazení. Vykresluje prázdné okno dle zadaných rozměrů a titulku. Hlavní komponenta pouze vytváří kontejner pro ostatní prvky uživatelského rozhraní. Při změně velikosti okna se volají příslušné C++ funkce, které uloží nové rozměry okna a vyvolají událost zaregistrovanou pomocí *on_resize()* ve třídě *window*.

ApplicationToolbar představuje část uživatelského rozhraní, které obsahuje menu a vypisuje titulek okna. Titulek je vykreslen především kvůli použití na embe-

[3] angl. *Observer*



Obrázek 5.6: QML komponenty

dded systému 2.1, kde se vykresluje do přímo framebufferu a není tak vidět žádné záhlavní okna. Komponenta umožňuje zaregistrovat zpětná volání, která se vyvolají při výběru položky menu. Obsluha zpětných volání probíhá až v C++ ve třídě *window* pomocí předem zaregistrovaných callbacků.

SingleModelScene je část grafického rozhraní, která vykresluje 3D vizualizaci pomocí knihovny WebGL. Zdrojový kód pro vykreslení objektů je napsán v jazyku JavaScript za podpory knihovny Three.js, která je dodána s knihovnou Qt 3.3. Komponenta je připravena pro vykreslení jednoho 3D modelu, který musí být načten z JSON souboru. Dále je obsažena kamera, která je nastavena do perspektivního režimu (obrázek 5.3). Vlastnosti 3D vizualizace (včetně kamery a vykreslovaného objektu) jsou ovládány pomocí parametrů předaných z C++ kódu.

Controls vytváří kontejner pro komponenty *ControlBox*. Komponenta *Controls* umožňuje posouvání („scrollování“) ovládacích prvků v případě, že není možné vykreslit všechny prvky do připravené oblasti.

ControlBox slučuje ovládací prvky do pojmenovaných skupin podle účelu, který prvky plní (například ovládání filtrů, uzamčení os,...). Komponenta očekává na svém vstupu název skupiny. Jednotlivé ovládací prvky jsou poté vykresleny jako potomci prvku *ControlBox*.

LabelSlider je komponenta tvořící ovládací prvek typu „range“ 5.4, ke kterému je navíc přidán popisek. V aplikaci diplomové práce je popisek použit pro zobrazení aktuální pozice ovládacího prvku. Při změně pozice z uživatelského rozhraní je

zavolaná C++ funkce, která obsluhuje danou změnu. Pro každou komponentu *LabelSlider* existuje v C++ samostatná instance třídy *range*, která vyvolanou událost obsluhuje.

Spinner představuje část uživatelského rozhraní, která indikuje zaneprázdněnost aplikace. Na obrázku 5.6 není komponenta zobrazena. Při zobrazení prvku *Spinner* přechází grafické rozhraní do deaktivovaného stavu (uživatelsky není možné zasahovat). Komponenta navíc zobrazuje popisek, který uživatele informuje o aktuální akci, která zapříčinila zaneprázdněnost aplikace.

AboutDialog podobně jako předchozí prvek není na obrázku 5.6 viditelný. Komponenta představuje vyskakovací okno, ve kterém jsou informace o aktuální aplikaci. Při otevřeném vyskakovacím okně dochází k deaktivaci zbytku uživatelského rozhraní, avšak na rozdíl od *Spinneru* je možné okno uživatelsky zavřít.

5.7 Konfigurační soubor

Konfigurovatelnost aplikace zaručuje její obecnější použití. Funkční blok *Konfigurační soubor* 3.4 představuje rozhraní pro práci s konfiguračními soubory. Podobně jako u implementace *Zařízení* a *Vizualizace* je knihovna rozdělena na třídy tvořící veřejná rozhraní a poté specifickou implementaci pro JSON konfigurační soubor. Ke složení rozhraní a implementace je opět použita metoda kompozice (skládání tříd) nikoli dědičnost 5.6.

Validator je třída tvořící rozhraní pro validaci dat na základě předem vytvořeného schématu. Validátor obsahuje metody, které umí zaregistrovat kontrolu konkrétních datových typů (číslo, řetězec, boleovská hodnota,…). Dále poskytuje rozhraní určující, zda jsou předaná data v předepsaném formátu. Rozhraní třídy je následující:

- *validator()* je konstruktor třídy, který pouze předává argumenty do metody *init()*.
- *init()* ukládá předané schéma. Zároveň je možné v konkrétních implementacích zaregistrovat primitivní datové typy a jejich validační funkce.
- *register_type()* zaregistruje nový typ nutný pro validaci dat (typ se vyskytuje ve schématu). Nový typ je zaregistrován dle předaného názvu typu a funkce, která zvaliduje data dle zadaných podmínek (příkladem může být název typu „positive_number“ a funkce, která navrátí „true“, pokud je testovaná hodnota kladná).
- *validate()* navrátí, zda předaná data plně vyhovují schématu určenému v metodě *init()*.

Config třída je hlavním rozhraním konfigurace. Předpokládá se, že bude inicializována již připraveným *validatorem*. Třída čte data z konfiguračního souboru a navrací je již ve správném datovém typu. Jestliže dojde k chybě načtení souboru, je použita vychozí konfigurace, která byla do třídy předána při inicializaci. Data je možné do konfiguračního souboru také zapisovat. Třída také poskytuje rozhraní pro zpětné uložení konfigurace do souboru. *Config* poskytuje veřejné rozhraní:

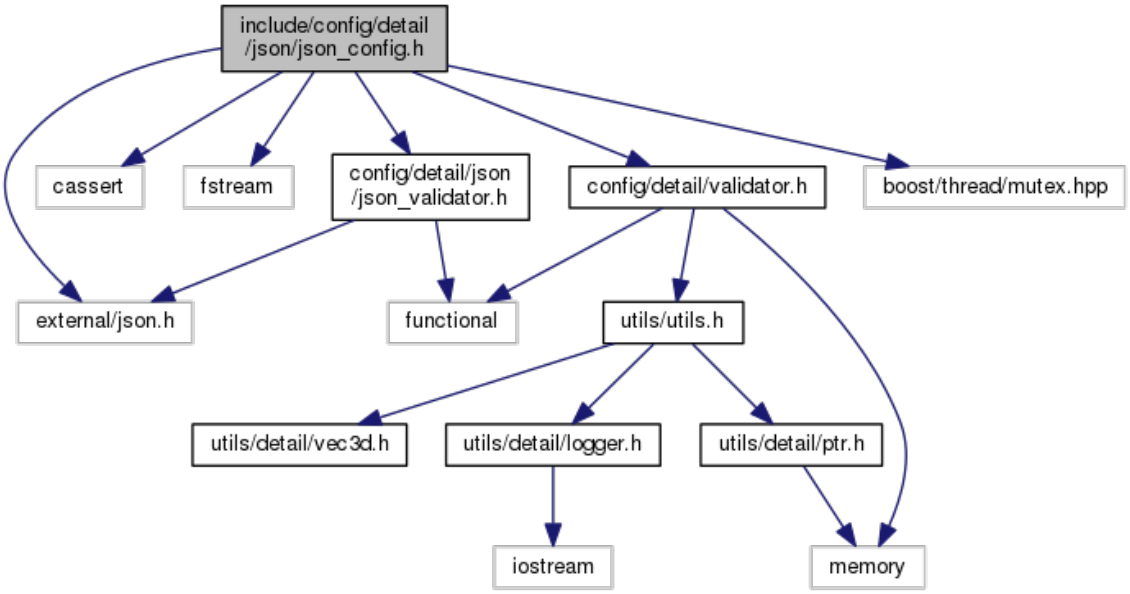
- *config()* je konstruktorem třídy konfigurace. Pouze volá metodu *init()* s předanými argumenty.
- *init()* nastaví konfiguraci do výchozí stavu. Na svém vstupu očekává cestu ke konfiguračnímu souboru, výchozí konfiguraci a již připravený *validator* (není možno jej měnit po nastavení třídy). Implementace rozhraní musí zkontrolovat platnost předané výchozí konfigurace pomocí *validatoru* a následně načíst konfigurační soubor. Jestliže dojde k chybě, je třída nastavena do chybového stavu.
- *read()* metoda navrací konkrétní datovou položku z konfigurace na základě cesty. Aby byla třída maximálně obecná, je cesta určena ve tvaru:

  ```
  cesta/ke/konkretni/hodnote
  ```

  Konkrétní cesta může představovat procházení skupin či různých návěští konfigurace. Jestliže datový typ načtené hodnoty nesouhlasí s typem požadovaným nebo hodnota není nalezena, je zavolána standardní funkce *assert* a program ukončen.
- *write()* poskytuje rozhraní pro změnu konkrétní hodnoty v konfiguračním souboru. Podobně jako u *read()* je konkrétní hodnota určena cestou. Po zapsání hodnoty musí dojít k revalidaci konfiguračního souboru. Jestliže dojde k chybě, je zavolána standardní funkce *assert* a program je ukončen.
- *save()* uloží konfigurační soubor držený v paměti počítače zpět do souboru. Pokud není zadána cesta k souboru, je původní soubor konfigurace nahrazen novým.
- *is_good()* otestuje, zda je třída správně nainicializována.
- *operator bool()* poskytuje příjemnější rozhraní pro testování stavu třídy.

5.7.1 JSON implementace

V budoucím vývoji je možné, že aplikace bude součástí většího celku. Aby bylo možné přizpůsibit formát konfigurace se zbytkem aplikace, poskytuje knihovna *Konfigurační soubor* pouze rozhraní, nikoli specifickou implementaci. Samotné chování pro konfiguraci formátu JSON implementují třídy *json_validator* a *json_config*. Třídy pracují s konkrétní implementací C++ JSON dle použité knihovny 3.3. Na obrázku 5.7 lze vidět diagram závislostí třídy *json_config*.



Obrázek 5.7: Diagram závislostí pro blok *Konfigurační soubor*

5.8 Centrální řídicí blok

Dle předchozího návrhu funkčních bloků 3.4 je patrné, že blok *Aplikace* tvoří centrální řídicí blok, který se stará o běh celého programu. Funkční blok řeší načítání dat z prostorového senzoru 2.2, jejich přepočet 2.3 a vizualizaci v uživatelském rozhraní. Řídicí blok již není naimplementován obecně, jelikož je to část zdrojového kódu, která je odvislá od chování konkrétní aplikace. *Aplikace* je tvořena pouze jednou třídou a statickými funkcemi pro zpracování dat ze senzorů.

Visualizer je hlavní třídou vizualizace dat 3D prostorového senzoru. Spojuje chování všech funkčních bloků, které byly popsány v předešlém textu. Hlavní činností je inicialiazace uživatelského rozhraní, konfiguračního souboru a třídy pro prostorový senzor. Dále poskytuje rozhraní pro spuštění a zastavení aplikace. Běh *visualizeru* je blokující. Je nutné mít na paměti, že třída nejde použít v jiném než hlavním vlákně, jelikož inicializuje uživatelské rozhraní, které musí vždy pracovat v hlavním vlákně. Třída ovšem obsluhuje asynchronní metodu vizualizační třídy *gui* a tím zajišťuje asynchronní aktualizaci uživatelského rozhraní. Rozhraní třídy je následující:

- *visualizer()* je konstruktor třídy. Pokud je zadán s prázdnými argumenty, vytvoří se pouze neinicializovaná třída. Pokud jsou zadány argumenty třídy (počet proměnných načtených z příkazové řádky a pole těchto proměnných), je zavolána metoda *init()*.
- *init()* inicializuje aplikaci. Nastaví a načte konfiguraci ze souboru, dále načte uživatelské rozhraní a inicializuje senzory a třídu *reader*, která bude vyčítat data ze zařízení. Třída používá Qt vizualizaci. Metoda dále zaregistruje potřebné callbacky.

- *run()* je blokující metoda, která spustí aplikaci.
- *stop()* zastavuje aplikaci.
- *is_good()* testuje, zda se aplikace správně nainicializovala.
- *operator bool()* pouze vytváří lepší rozhraní pro testování, zda je je aplikace správně nainicializována.

Třída *visualizer* dále obsluhuje callbacky třídy *gui*. První zpětné volání *interface_async_handler()* slouží jako pracovní vlákno. V nekonečném cyklu načítá data ze senzorů, aplikuje na načtené hodnoty metody pro zpracování dat 2.3 a na základě vypočtených hodnot aktualizuje uživatelské rozhraní. Druhý callback *on_interface_opened()* je zavolán jakmile se otevře nové uživatelské rozhraní. Slouží k nastavení a inicializaci grafických prvků, které se v rozhraní vyskytují.

Pro větší obecnost a efektivitu kódu jsou statické funkce pro zpracování dat napsané samostatně a navíc tzv. „inline“ (při překladu je volání inline funkce nahrazeno kódem, který funkce zpracovává – zpracování kódu je rychlejší). Dostupné funkce pro zpracování dat jsou:

- *low_pass()* je funkce pro dolnopropustní filtr (rovnice 2.1). Výstupní hodnotou je nově vypočtená hodnota z filtru.
- *high_pass()* provede nad daty hornopropustní filtraci dle rovnice 2.6. Výstupem je vyfiltrovaná hodnota.
- *complementary()* provede fúzi dat 2.12 dvou senzorů. Na výstupu je hodnota spočítaná filtrem.
- *to_radians()* převede úhel ve stupních na radiány a navrátí vypočtenou hodnotu.
- *to_degrees()* navrátí úhel přepočítaný z radiánů na stupně.
- *to_zero2pi()* převádí úhel z intervalu $\langle -\pi, \pi)$ na úhel z intervalu $\langle 0, 2\pi)$.
- *move_angle()* posouvá úhel o zadaný offset, avšak zachovává obor hodnot $\langle -\pi, \pi)$.
- *get_pitch()* spočítá úhel *pitch* dle rovnice 2.2 a opraví jej, aby byl zachován obor hodnot $\langle -\pi, \pi)$ 2.4.
- *get_roll()* vypočítá z dat akcelerometru úhel *roll* pomocí rovnice 2.3. Následně úhel opraví jako v předešlém případě podle vzorce 2.5.
- *get_yaw()* vypočítá úhel *yaw* z dat magnetometru 2.11.

6 Testování

Testování aplikace patří mezi nejdůležitější kroky při vývoji nového software. Vyvíjený systém se musí testovat již během fáze návrhu 3, kde jsou testy zaměřené na jednotlivé technologie (embedded systém, knihovny). Během samotné implementace je poté nutné testovat jednotlivé funkční části zdrojových kódů (jednotkové testy, náhodné vstupy,...). Postupným testováním vyvíjeného systému se předejde chybám a rizikům, které by mohly z návrhu či implementace později vyplynout. V poslední řadě se software testuje jako celek.

Během tvorby aplikace a testování se detekovaly chyby se zpracováním dat. U rotací dochází ke správnému přepočtu dat senzorů ve chvíli, kdy se rotuje okolo každé osy zvlášť (pouze jedna osa je „odemčená“). Jakmile se povolí rotace kolem osy *pitch* a *roll* dohromady, pracují přepočty správně jen v rozmezí $\langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$. Popsaný problém souvisí se změnou znaménka gravitační síly v hraničních úhlech (neboli problém závisí na výstupu akcelerometru). Bohužel ani po konzultacích s panem doc. Ing. Pavlem Rydlem, Ph.D nebyl zmíněný problém vyřešen.

K objektivnějšímu zhodnocení aplikace byly provedeny uživatelské testy. Popis scénáře testů a následné zhodnocení testování je popsané v následujících sekcích. Zároveň druhá část kapitoly obsahuje doporučení pro další vývoj.

6.1 Popis testování

K otestování aplikace byla vybrána skupina čtyř studentů, kteří docházejí na předmět „Minipočítače a jejich praktické aplikace“. Do odpovědí bylo také zahrnuto zhodnocení aplikace od vyučující paní Ing. Lenky Koskové Třískové. Připravené úkoly měly ověřit citlivost ovládání, zpětnou odezvu a interakci s uživatelským rozhraním. Každému uživateli byly poskytnuté informace k ovládání aplikace a všichni byli informováni o předchozí popsané chybě.

Scénář testování

Aplikace byla zprovozněna a načtena do svého původní stavu. Úkolem uživatelů bylo:

- Vyzkoušet citlivost a přesnost prostorového senzoru a zároveň ověřit rychlou zpětnou odezvu vizualizace.

- Otestovat ovládací prvky a změnit nastavení uživatelského rozhraní za pomoci myši či klávesnice.

Po testování byla každému uživateli předložena anonymní online anketa. Anketa obsahovala otázky směřovala otázky na citlivost a odezvu prostorového senzoru, intuitivnost uživatelského rozhraní a zároveň umožňovala připsat připomínky k danému řešení či obodovat aplikaci dle celkového dojmu z ovládání.

6.2 Výsledky a zhodnocení testování

Po získání odpovědí od všech uživatelů byly výsledky jednotlivých otázek vyhodnoceny následujícím způsobem (na otázku celkového bodového ohodnocení odpověděli všichni uživatele maximálním počtem bodů):

- *Je ovládání intuitivní a vhodné pro dotykové displeje?* Všichni uživatelé bez výjimky odpověděli na otázku pozitivně.
- *Podařilo se Vám splnit všechny zadané úkoly?* Všem uživatelům se podařilo splnit zadané úkoly bez problému.
- *Popište problém s ovládáním, pokud se nějaký vyskytl.* Otázka nebyla povinná a odpověděli pouze dva uživatele. První uživatel popsal problém s rotacemi *pitch* a *roll*, který byl v předchozím textu již popsán. Druhý uživatel měl problém se „synchronizací“ rotace prostorového senzoru a vykreslované rotace 3D modelu – na první pohled nejsou zřejmé osy senzoru a vizualizovaného objektu.
- *Jak citlivý byl prostorový senzor?* Čtyři z pěti testovaných uživatelů odpověděli, že citlivost senzoru a velice přesná. Jeden z testovaných podotknul, že při malém otočení se vizualizovaný objekt otočí více než senzor.
- *Přišla Vám odezva vizualizace dostatečně rychlá?* Všichni uživatelé se shodli, že odezva aplikace je dostatečně rychlá.
- *Jaký je Váš celkový dojem z aplikace? Co se Vám líbilo, nebo naopak nelíbilo?* Celkový dojem všech uživatelů z aplikace byl kladný a aplikace byla ohodnocena velmi pozitivně. Jeden z uživatelů navrhnul zobrazení os ve 3D vizualizaci a u prostorového senzoru.

Největším problémem byla synchronizace prostorového senzoru a vizualizace. Jelikož prostorový senzor (stejně jako vizualizace) nemá označené osy, je pro uživatele náročnější srovnat polohu senzoru a zobrazovaného 3D modelu. V dalším vývoji by bylo vhodné osy vizualizovat. Pro prostorový senzor bude nutné v budoucnu navrhnout pouzdro, které bude vhodné pro odvětví rehabilitace. Při návrhu pouzdra bude nutné myslet na jednoduché určení orientace senzoru.

Do dalšího vývoje bude potřebné opravit problém se zpracováním dat v rotacích *pitch* a *roll*, aby bylo možné zařízení plynule otáčet při „odemčení“ všech os.

Výsledek testování byl ovšem pozitivní. Všichni uživatelé splnily zadané úkoly bez problému a ovládání aplikace je zhodnoceno jako intuitivní a připravené pro dotykové ovládání. Zároveň byl velice kladně hodnocen prostorový senzor, který byl pro uživatele citlivý a vizualizace měla rychlou odezvu na změny rotace senzoru.

Ač je aplikace diplomové práce pouze prototypem, poznatky z provedeného testování ukázaly, že vývoj směřuje správným směrem.

7 Závěr

Tématem práce bylo seznámení se s předloženým embedded systémem a způsobem vykreslení 3D prostorových souřadnic na zadaném systému a dále s prostorovým senzorem používáným pro rehabilitace a formátem dat, které senzor poskytuje. Hlavním cílem bylo navrhnout a implementovat prototyp aplikace pro vizualizaci dat 3D prostorového senzoru. Navržené řešení mělo být dostatečné obecné a především snadno do budoucna rozšířitelné.

Hlavní záměr praktické části byl splněn, tudíž byl vytvořen funčkní prototyp, který vizualizuje data 3D prostorového senzoru v embedded systému. Jelikož návrh vlastního embedded systému by byl časově i finančně nákladný, byl použit minipočítač Raspberry Pi, který simuloval činnost embedded systému. Z předložených prostorových senzorů se pro práci použil senzor LSM9DS0. Veškeré zdrojové kódy jsou rozděleny na rozhraní a implementaci, tudíž je řešení dostatečně obecné a především dobře rozšířitelné v budoucím vývoji. Následující uživatelské testování odhalilo nedostatky aplikace především při zobrazování rotace. Aplikace umí pracovat správně pouze v intervalu úhlů $\left\langle -\frac{\pi}{2}, \frac{\pi}{2} \right\rangle$. Zároveň bylo pro uživatele složité synchronizovat počáteční orientaci prostorového senzoru s vizualizovaným 3D modelem. Avšak i přes nedostatky aplikace bylo ovládání zhodnoceno jako inutitivní a celkový dojem z aplikace byl kladný. Senzory jsou velice citlivé a přesné a pro využití v rehabilitacích vhodné. Do dalšího vývoje bude nutné opravit chybu se zobrazováním rotace a dále navrhnout vlastní embedded systém a celkové řešení pro prostorový senzor.

Reference

[1] Lee, E. A.; Seshia, S. A.: *Introduction to embedded systems: a cyber-physical systems approach*. Morrisville, NC: LeeSeshia.org, 2011, ISBN 9780557708574, 11-12 s., [cit. 2017-02-12].

[2] Kamal, R.: *Examples of embedded systems [online]*. State university of Madhya Pradesh, India, 2008, [cit. 2017-02-21]. Dostupné z: http://www.dauniv.ac.in/downloads/EmbsysRevEd_PPTs/Chap01Lesson_6Emsys.pdf

[3] Kosková-Třísková, L.: *Minipočítače, Úvod*. Fakulta mechatroniky, informatiky a mezioborových studií, A:A3, Letní semestr, 2017, [cit. 2017-03-01].

[4] Wikimedia Foundation, Inc.: *Comparison of single-board computers*. 2017, [cit. 2017-03-01]. Dostupné z: https://en.wikipedia.org/wiki/Comparison_of_single-board_computers

[5] Acme Systems srl: *Acqua A5 - System On Module [online]*. 2017, [cit. 2017-02-22]. Dostupné z: https://www.acmesystems.it/acqua

[6] Acme Systems srl: *BERTA-A5 - LCD Evaluation board [online]*. 2017, [cit. 2017-02-22]. Dostupné z: https://www.acmesystems.it/acqua

[7] Software in the Public Interest, Inc.: *Debian - The Universal Operating System [online]*. 2016, [cit. 2017-02-22]. Dostupné z: https://www.debian.org/

[8] Khronos Group: *OpenGL - The Industry's Foundation for High Performance Graphics [online]*. 2017, [cit. 2017-02-22]. Dostupné z: https://www.opengl.org/

[9] Raspberry Pi Foundation: *Raspberry Pi 1 Model B+ [online]*. 2017, [cit. 2017-02-22]. Dostupné z: https://www.raspberrypi.org/products/model-b-plus/

[10] Raspberry Pi Foundation: *Raspberry Pi 3 Model B [online]*. 2017, [cit. 2017-02-22]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[11] Mitchell, H. B.: *Multi-Sensor Data Fusion: An Introduction*. Springer, 2011, ISBN 9783540714637, 3-7 s., [cit. 2017-02-18].

[12] Wolfram Research, Inc: *Euler Angles*. 2017, [cit. 2017-03-10]. Dostupné z: http://mathworld.wolfram.com/EulerAngles.html

[13] GiTy, a.s.: *Komplexní řešení IT a komunikací [online]*. 2015, [cit. 2017-03-10]. Dostupné z: http://www.gity.cz/

[14] Zajacová, P.: *Využití akcelerometrických senzorů v rehabilitaci*. Liberec: Technická univerzita, Ústav zdravotnických studií, 2014, vedoucí diplomové práce Ing. Martin Kysela.

[15] Bluetooth SIG, Inc: *Bluetooth*. 2017, [cit. 2017-03-10]. Dostupné z: https://www.bluetooth.com/

[16] Analog Devices, Inc.: *3-Axis Digital accelerometer*. 2017, [cit. 2017-03-10]. Dostupné z: http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[17] InvenSense, Inc.: *3-Axis Gyroscope*. 2017, [cit. 2017-03-10]. Dostupné z: https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf

[18] STMicroelectronics: *LSM9DS0 [online]*. 2017, [cit. 2017-03-10]. Dostupné z: http://www.st.com/en/mems-and-sensors/lsm9ds0.html

[19] Kolář, M.: *Sběrnice*. Fakulta mechatroniky, informatiky a mezioborových studií, A:TK10, Zimní semestr, 2015, [cit. 2017-03-10].

[20] Lahorde: *Linux driver for lsm9ds0 IMU [online]*. GitHub, Inc, 2017, [cit. 2017-03-10]. Dostupné z: https://github.com/Lahorde/st_lsm9ds0_linux_driver/

[21] Málek, J.: *Číslicové filtry*. Fakulta mechatroniky, informatiky a mezioborových studií, B:B9, Letní semestr, 2016, [cit. 2017-03-15].

[22] Pedley, M.: *Tilt Sensing Using a Three-Axis Accelerometer*. Freescale semiconductor, Inc, 2013, [cit. 2017-03-19]. Dostupné z: https://www.nxp.com/files/sensors/doc/app_note/AN3461.pdf

[23] IBM Corporation: *ATAN2(Y,X) [online]*. 2003, [cit. 2017-03-16]. Dostupné z: http://www.msg.ucsf.edu/local/programs/IBM_Compilers/Fortran/html/pgs/lr202.htm

[24] Reichl, J.; Všetička, M.: *Pohyb hmotného bodu po kružnici*. 2017, [cit. 2017-03-16]. Dostupné z: http://fyzika.jreichl.com/main.article/view/15-pohyb-hmotneho-bodu-po-kruznici

[25] Woodman, O. J.: *An introduction to inertial navigation*. Cambridge, United Kingdom, August 2007, [cit. 2017-03-16]. Dostupné z: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf

[26] Ozyagcilar, T.: *Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors*. Freescale semiconductor, Inc, 2013, [cit. 2017-03-19]. Dostupné z: https://cache.freescale.com/files/sensors/doc/app_note/AN4248.pdf

[27] Welch, G.; Bishop, G.: *An Introduction to the Kalman Filter*. University of North Carolina at Chapel Hill, 1997, [cit. 2017-03-17]. Dostupné z: http://tinyurl.com/kalmanfilter2017

[28] Colton, S.: *The Balance Filter*. Massachusetts Institute of Technology, June 25, 2007, [cit. 2017-03-17]. Dostupné z: http://d1.amobbs.com/bbs_upload782111/files_44/ourdev_665531S2JZG6.pdf

[29] The Qt Company, Ltd.: *Qt for Embedded Linux [online]*. 2017, [cit. 2017-04-25]. Dostupné z: http://doc.qt.io/qt-5/embedded-linux.html

[30] Høgsberg, K.: *Wayland Architecture*. 2017, [cit. 2017-03-25]. Dostupné z: https://wayland.freedesktop.org/

[31] ARM, Ltd.: *ARM Markets Mobile [online]*. 2017, [cit. 2017-03-30]. Dostupné z: https://www.arm.com/markets/mobile

[32] Software in the Public Interest, Inc.: *The Computer Language Benchmarks Game [online]*. 2017, [cit. 2017-03-30]. Dostupné z: http://benchmarksgame.alioth.debian.org/

[33] Pihlajamaa, J.: *Benchmarking Raspberry Pi GPIO Speed [online]*. 2017, [cit. 2017-03-30]. Dostupné z: http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/

[34] (nlohmann), N. L.: *JSON for Modern C++ [online]*. GitHub, Inc., 2017, [cit. 2016-05-02]. Dostupné z: https://github.com/nlohmann/json

[35] ROLI Ltd.: *JUCE [online]*. 2017, [cit. 2017-04-04]. Dostupné z: https://www.juce.com/

[36] The GTK+ Team: *The GTK+ Project [online]*. 2017, [cit. 2017-04-04]. Dostupné z: https://www.gtk.org/

[37] The Qt Company, Ltd.: *Cross-platform software development for embedded & desktop [online]*. 2017, [cit. 2017-04-04]. Dostupné z: https://www.qt.io/

[38] The Ohio State University: *Model-View-Controller (MVC) Design Pattern [online]*. 2017, [cit. 2017-04-16]. Dostupné z: http://web.cse.ohio-state.edu/~rountev.1/421/lectures/lecture23.pdf

[39] Katedra počítačové grafiky a interakce, Fakulta elektrotechnická, České vysoké učení technické: *Jak psát komentáře v Doxygenu [online]*. 2014, [cit. 2017-04-20]. Dostupné z: https://cent.felk.cvut.cz/courses/PGR/doxygen.html

[40] Open Source Initiative: *Licenses & Standards*. 2017, [cit. 2017-04-12]. Dostupné z: https://opensource.org/licenses

A Obsah přiloženého CD

- `readme.txt` — stručný popis obsahu CD
- `src/` — zdrojové soubory
  - `application/` — zdrojové kódy aplikace
    - `config/` — složka obsahující konfigurační soubory
      - `qt_lsm9ds0.json` — konfigurace pro senzor LSM9DS0 a knihovnu Qt
    - `doc/` — adresář obsahující vygenerovanou dokumentaci
      - `html/` — dokumentace ve formátu HTML
    - `include/` — hlavičkové soubory aplikace
      - `config/` — definice funkčního bloku konfigurace
      - `device/` — definice funkčního bloku zařízení
      - `error/` — hlavičkové soubory pro chyby
      - `external/` — externí knihovna pro JSON
      - `utils/` — hlavičkové soubory utilit
      - `view/` — definice funkčního bloku vizualizace
    - `src/` — zdrojové C++ soubory aplikace
      - `config/` — funkční blok konfigurace
      - `device/` — funkční blok zařízení
      - `error/` — pomocné soubory pro odchycení chyb
      - `utils/` — vlastní utility používané v aplikace
      - `view/` — funkční blok vizualizace
      - `main.cxx` — hlavní zdrojový soubor aplikace
    - `ui/` — soubory uživatelského rozhraní QML
    - `Makefile` — řídicí soubor kompilace
    - `Doxyfile` — řídicí soubor pro generování dokumentace
    - `README.md` — soubor obsahující základní informace k instalaci
  - `thesis/` — zdrojová forma práce ve formátu LaTeX
    - `assets/` — externí soubory připojené k práci
      - `images/` — obrázky použité v diplomové práci
      - `zadani.pdf` — originální zadání
- `text/` — text práce
  - `dip_lukas_sedlacek.pdf` — text práce ve formátu PDF